Generating Tetrahedral Mesh for Solving Ion Channel Simulation Problems Using Finite Element Methods - REU Report

Yueyang Ding, Mentor: Zhen Chao

July 7, 2023

Abstract

This paper presents a thorough and effective strategy for the construction of ion channel protein tetrahedral meshes.

When first construct a surface mesh for the ion channel -membrane structure. We transformed the contours of box surface meshes and incorporated membrane surface mesh points. These steps help shape tetrahedrons in the membrane region appropriately. We union such box surface with the membrane's Gaussian surface constructed by the *TMSMesh* software to get the surface mesh of the whole structure. Next, we employ volume mesh generation software, particularly *Tetgen*, to establish volume meshes. Afterwards, a crucial part of this project is extracting the membrane and solvent labels in the membrane area, which we achieve through graph search algorithms. In our study, we introduce three distinct yet related methods: Breadth First Search (BFS), Two-Step BFS, and Two-Step BFS utilizing straight walk.

To effectively illustrate our approach, we provide a visual representation of the tetrahedral mesh generated for three exemplary ion channel proteins.

Our approach, implemented in C++, serves as a valuable tool for addressing ion channel simulation problems using the finite element method. This research was conducted as part of the 2023 Summer REU program at the University of Michigan.

1 Introduction

Different kinds of ions are an integral part of the microscopic cellular environment. For instance, sodium, potassium, calcium, and chloride play an important role in many different physiological processes, including hormone secretion, cell communication, muscle contraction, and neuron signaling. [6] Ion channels are pore-forming proteins found in the plasma membrane. They control the small voltage gradient across the membrane, allowing ions to pass through. Different types of ion channels are involved in various crucial physiological processes. Studying ion channels and their behavior is important for understanding a wide range of biological functions and developing treatments for various diseases.

While many models exist for the ion channel simulation problem, PBE (Poisson-Boltzmann equation) and PNP (Poisson-Nernst-Planck) equations are two widely used dielectric continuum models. When solving those equations using numerical methods like finite element methods, a high quality tetrahedral mesh is required as a discrete representation of the domain, the ion channel protein embedded in the membrane and immersed in some electrolyte fluid. For example, when solving the following Poission dielectric equations, the domain should be partitioned into that three parts, protein, membrane and solvent, corresponding to D_p, D_m, D_s where $u(\mathbf{r})$ represents the [3]

$$
\begin{aligned}
-\epsilon_p \Delta u(\mathbf{r}) &= \alpha \sum_{j=1}^{n_p} z_j \delta_{r_j}, r \in D_p, \\
-\epsilon_m \Delta u(\mathbf{r}) &= 0, r \in D_m, \\
-\epsilon_s \Delta u(\mathbf{r}) &= \beta \sum_{i=1}^{n} Z_i c_i(\mathbf{r}), \mathbf{r} \in D_s,
\end{aligned}
$$

There already exists many software regarding mesh generation of ion channels and labeling that into three partitions. For Instance, the ICMPv1 and ICMPv2 [2] and also the algorithms presented by Liu et al. which have utilize the straight walk algorithms during the membrane extraction stage. [5, 1] Most of those applications varies in their methods of surface mesh generation and the membrane/solvent tetrahedral extraction. In this research project, we aim to propose some algorithms with an emphasis on efficiency. While previous methods may somehow rely on packages in python, we want to implement them using mostly C++ which might be improving the cpu run time.

Generally, our methods would adopt the framework of ICMPv2 [2] and we will provide three different approaches for membrane meshes extraction, including plain breath first search, two-step breath first search, and two-step breath first search with the straight walk algorithm. Besides that, we follow the framework of ICMPv2 on deciding the box domain, and generating surface meshes. Two softwares were used as tools in our software, *TMSMesh* [4] and *TetGen* [7] with efficient functions of generating the Gaussian Surface triangular mesh and volume mesh, respectively. Other substitutes for protein surface generating software are *NanoShaper* or *MSMS*.

2 Surface Mesh Generation of Surface Box

Modeling the specific structure of the membrane would be too complicated and not realistic. So we define a simplified model to place the membrane between the plane $\{(x, y, z) \in \Omega \mid z = Z_1\}$ and $\{(x, y, z) \in \Omega \mid z = Z_2\}$ while Z_1 and Z_2 are the locations of membrane determined by the specific ion channel protein. During this research, we use the protein PDB files from the *OPM Database* [1] The proteins in the database are well-organized with the membrane location Z_1, Z_2 symmetric around $Z = 0$ by the value indicated as *1/2 of bilayer thickness* in the PDB file.

We define $\Gamma_{s,h}$ as the surface triangulation on the four sides and $\Gamma_{tb,h}$ as the surface triangulation on the top and bottom. The entire surface mesh can thus be expressed as

$$\Gamma_h = \Gamma_{s,h} \cup \Gamma_{tb,h}$$

The membrane region is represented by a denser segment of the surface triangular mesh on Γ_s, h. This configuration ensures that the membrane location information is integrated into the final tetrahedral mesh while simultaneously satisfying the precision requirements of ion channel simulation. Between the membrane area and the margin area, there is a connecting area where we connect mesh points of different densities in a straightforward way. To simplify this connecting area, in our experimental implementation, we let the surface mesh density in the membrane area be double that in the margin area. i.e. Assume the surface mesh grids on the margin area have a side length of H_S while those surface mesh grids at the membrane area have a side length H_M, we assume that:

$$H_S = 2 \cdot H_M$$

Note that:

$$\Gamma_{tb,h}^{Membrane} = \{\mathbf{r} \in \Gamma_{tb,h} \mid \mathbf{r} = (x, y, z) \text{ with } Z_1 \leq z \leq Z_2\}$$
$$\Gamma_{tb,h}^{Margin} \cup \Gamma_{tb,h}^{Connecting} = \{\mathbf{r} \in \Gamma_{tb,h} \mid \mathbf{r} = (x, y, z) \text{ with } z < Z_1\} \cup \{\mathbf{r} \in D_{s,h} \mid \mathbf{r} = (x, y, z) \text{ with } z > Z_2\}$$

More specifically, our surface mesh setting is the same as the setting used in ICMPV2 [2]

The top and bottom of the structure involve standard surface triangulation with mesh grids of side length H_S. This surface triangulation is represented in the *poly* format, specifically designed for the volume mesh generation software, TetGen [7]. More precisely, we compose a list of vertices, each followed by the components of their respective position vectors. Additionally, we include a list of triangle components, each followed by the indices of their three vertices.

We have included this figure to provide a visual example of the surface mesh that was established earlier in the paper. The figure is based on the poly file generated through a sample triangular mesh that we created. We have used the python library *pyplot* to create this figure, which is a widely-used tool for data visualization.

[1]the University of Michigan Orientations of Proteins in Membranes (OPM) database; https://opm.phar.umich.edu/

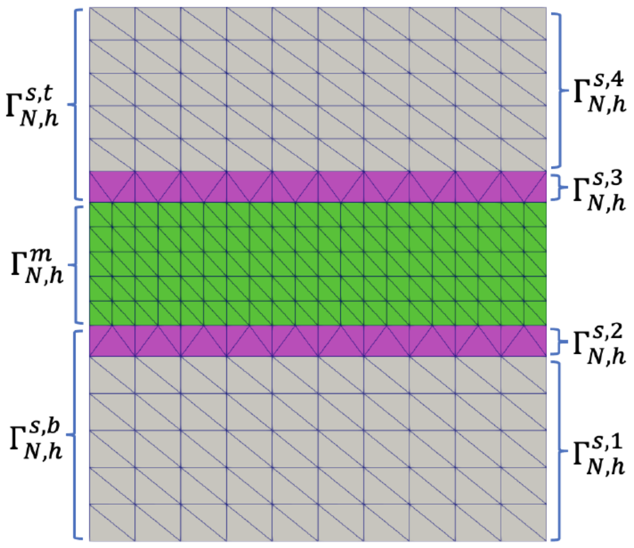


Figure 1: Surface triangular mesh partition on the sides used in ICMPV2 [2]

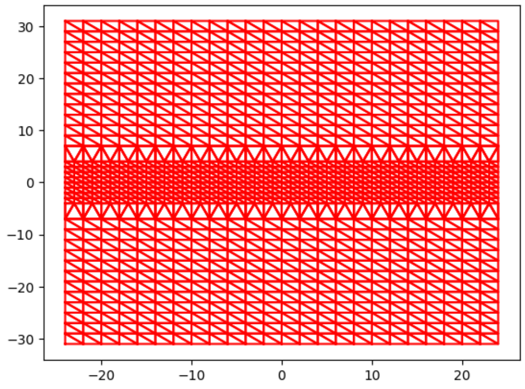


Figure 2: Visualisation of a sample box surface mesh

3 Surface Mesh Generation of Protein

In order to generate tetrahedral meshes using volume mesh generation software like *Tetgen* [7], a surface mesh is required as input. We define the domain of our tetrahedral mesh, $\Omega \in \mathbb{R}^3$, as an ion channel protein situated within a box. Here, $\partial\Omega$ represents the surface mesh for the surface box, Γ_h, which we generated in the previous section. Additionally, we require the surface triangulation for the protein embedded within the box, $\partial\mathcal{P}$. This is obtained via a triangular surface mesh of a molecular surface. In this research project, we choose to employ a Gaussian surface, and the software *TMSMesh* [4] to generate such surface triangulation from the *PQR file* that outlines the molecular structure of a protein.

While there are several other molecular surfaces available, such as the van der Waals surface (VDW) - which is the union of each molecule's spherical domain with a certain radius - as well as solvent-accessible surfaces (SAS) and solvent-excluded surfaces (SES) - both obtained from the path of a spherical probe traversing the VDW surface, we opt to utilize the Gaussian surface in this research project. The Gaussian surface, with its smooth characteristics, is especially suited for representing a molecule's electron density.

3.1 Gaussian Surface

Gaussian Molecular surface is defined as a level set of the sum of Gaussian kernel functions. [4] Where the definition can be mathematically expressed as:

$$\left\{\vec{x} \in R^3, \phi(\vec{x}) = c\right\}$$

$$\phi(\vec{x}) = \sum_{i=1}^{N} e^{-d\left(\|\vec{x}-\vec{x}_i\|^2 - r_i^2\right)}$$

In these equations, $\vec{x}_i$ and r_i represent the location vector and radius of the i-th atom, respectively. The parameter d denotes the decay rate for the Gaussian surface. As the decay rate decreases, the resulting surface becomes smoother and more inflated. The constant c is the isovalue that defines the Gaussian surface as a level set, and it governs the volume enclosed by the surface.

3.2 Using ***TMSMesh*** [4]

TMSMesh gives us a triangular surface mesh of the Gaussian surface of a certain molecular structure. It uses approximation and numerical method to calculate the Gaussian surface and then use a specific algorithm to generate triangles based on points on the Gaussian surface. The software takes a *pqr file* and three parameters h, d, c as input.

Here is a sample of a *PQR file*, it encodes information of a large molecular structure while each row corresponds to an atom. It contains multiple features of atoms including atom name, residue name, residue number, components of it's position vector, charge and radius. *TMSMesh* will utilize the position vector and radius when constructing triangular surface mesh.

Table 1: Exmaple of PQR File of the protein *1mag*

1	N	ALA	23	-16.378	6.310	-19.289	-0.3000	1.8500
2	CA	ALA	23	-15.241	5.381	-19.555	0.2100	2.2750
3	C	ALA	23	-15.333	4.149	-18.663	0.5100	2.0000
4	O	ALA	23	-16.317	3.409	-18.705	-0.5100	1.7000
5	CB	ALA	23	-13.914	6.100	-19.317	-0.2700	2.0600
6	H	ALA	23	-17.218	5.781	-19.200	0.3300	0.2245

TMSMesh takes in three parameters, h, d, c. d, c are the same parameters defined in the Gaussian Surface definition, while h relates to the triangulated mesh's density. It generates the triangular surface mesh of the protein $\partial\mathcal{P}$ also in the *poly* form.

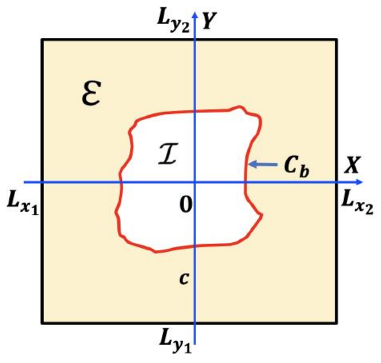
4 Add Membrane Surface Meshpoints

To more accurately simulate the structure of an ion channel embedded in a membrane, we need to refine our approach. In addition to densifying the membrane segment on the surface of the box, we aim to incorporate mesh points within the meshes on the upper and lower surfaces of the membrane inside the surface box. However, it's important that these points do not intrude into the region enclosed by the protein. This procedure ensures that the tetrahedral mesh associated with the membrane is planar and well-shaped along the margins.

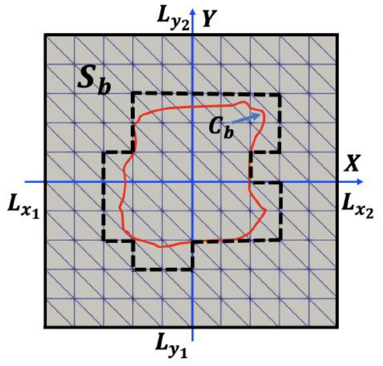
Following the idea of membrane surface mesh point used in ICMPV2 [2] We numerically define this set of mesh points on the bottom surface of membrane S_b in the following way. The upper surface does a similar way.

$$\mathcal{T} = \{(x_i, y_j) \mid x_i = L_{x_1} + ih_m, y_j = L_{y_1} + jh_m \text{ for } i = 0, 1, \ldots, m, j = 0, 1, \ldots, n\}$$
$$S_b = \{(x_i, y_i, Z_1) \mid (x_i, y_i) \in \mathcal{T} \cap \mathcal{E}\}$$

To find this set using an algorithm, we defined a numerical scheme, a lower and upper bound those protein surface mesh points at the cross-section $z = Z_1$ for x or y given y or x. That is:



(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 3: Visualisation of membrane surface mesh points from [2]

$$\begin{array}{ll}
L_y^1(x_i) = \min\{y_i \mid (x_i, y_i, Z_1) \in \mathcal{P}\} & U_y^1(x_i) = \max\{y_i \mid (x_i, y_i, Z_1) \in \mathcal{P}\} \\
L_x^1(y_i) = \min\{x_i \mid (x_i, y_i, Z_1) \in \mathcal{P}\} & U_x^1(y_i) = \max\{x_i \mid (x_i, y_i, Z_1) \in \mathcal{P}\} \\
L_y^2(x_i) = \min\{y_i \mid (x_i, y_i, Z_2) \in \mathcal{P}\} & U_y^2(x_i) = \max\{y_i \mid (x_i, y_i, Z_2) \in \mathcal{P}\} \\
L_x^2(y_i) = \min\{x_i \mid (x_i, y_i, Z_2) \in \mathcal{P}\} & U_x^2(y_i) = \max\{x_i \mid (x_i, y_i, Z_2) \in \mathcal{P}\}
\end{array}$$

Hereby we conclude:

$$\{(x_i, y_i) \mid x_i \notin (L_x(y_i), U_x(y_i))\} \cup \{(x_i, y_i) \mid y_i \notin (L_y(x_i), U_y(x_i))\} \subset \mathcal{E}$$

To translate this concept into an algorithm, we continuously track the lower and upper bounds in a discrete manner using an array. This is achieved by iterating over all points in the cross-section of all points in the protein's surface mesh - that is, all vertices in the *poly* file obtained from the *TMSMesh* step. Subsequently, we iterate over all potential mesh points, determining whether they should be incorporated into the meshes using the expression above. The specific algorithm is defined here.

5 Volume Mesh Generation

The union $\partial\Omega \cup \partial\mathcal{P} \cup S_b$ provides us with the surface mesh of the protein-membrane structure we aim to simulate. Assuming the protein is well-situated within the membrane section, we can treat this surface mesh as a boundary representation of a piece-wise linear complex (PLC), given that it does not include internal intersections. This PLC boundary representation can be fed into the *Tetgen* software, which in turn will generate a high-quality volume mesh. More specifically, it will produce a Constrained Delaunay Tetrahedralization of this PLC structure. Delaunay property could make sure the complexes tend to be distributed evenly through out the domain while keeping the boundary to remain a certain structure. [7]

By adjusting certain parameters and supplying the surface mesh *poly* file as input, *Tetgen* generates high-quality tetrahedral meshes. These are accompanied by a *nodes* file, which contains the position of each node; an *ele* file, which lists the components (i.e., indices of the four nodes) of each tetrahedron; and a *neigh* file, which includes the index of up to four neighbors for each tetrahedron. Additionally, the *ele* file assigns a label to each tetrahedron, corresponding to the region to which the tetrahedron belongs.

Since our surface mesh, as a PLC, demarcates two regions—namely, the protein and the non-protein region within the box (encompassing solvent and membrane)—we need to ensure that these regions are correctly labeled. Thus, we specify to *Tetgen* which label corresponds to a particular region. To accomplish this, we identify a point $p \in \mathbb{R}^3$ that we know resides in the solvent and membrane region, and assign a specific label to that region, i.e., add that point in the region section of the *poly* input file with a specific label. In this project, we set p as $(P_x + \epsilon, 0, 0)$, where P_x is the maximum

Algorithm 1 Add Membrane Surface Mesh Points

```
1: procedure ADD_SURFACE_MESH_POINTS
2:     Initialize $L_x^{Z1}, U_x^{Z1}, L_y^{Z1}, U_y^{Z1}, L_x^{Z2}, U_x^{Z2}, L_y^{Z2}, U_y^{Z2}$, as 1D array with boundary value
3:     Parse $protein\_num\_nodes, protein\_num\_triangles$ from reading $protein\_off\_file$
4:     Define $\mathcal{Z}_1 = (Z_1 - \epsilon, Z_1 + \epsilon), \quad \mathcal{Z}_2 = (Z_2 - \epsilon, Z_2 + \epsilon)$
5:     for each $i$ from 0 to $|\mathcal{N}|$ do
6:         Read line from $\mathcal{N}$ and parse it to get $x, y, z$
7:         if $z \in \mathcal{Z}_1 \cup \mathcal{Z}_2$ then
8:             Assume $z \in \mathcal{Z}_\alpha$ for $\alpha \in \{1, 2\}$
9:             $x_i \leftarrow (x + \frac{H_G}{2} - L_{x1})/H_G$
10:            $y_i \leftarrow (y + \frac{H_G}{2} - L_{y1})/H_G$
11:            Update $L_x^{Z\alpha}[y_i] \leftarrow \min(L_x^{Z\alpha}[y_i], x_i) \quad U_x^{Z\alpha}[y_i] \leftarrow \max(U_x^{Z\alpha}[y_i], x_i)$
12:            Update $L_y^{Z\alpha}[x_i] \leftarrow \min(L_y^{Z\alpha}[x_i], y_i) \quad U_y^{Z\alpha}[x_i] \leftarrow \max(U_y^{Z\alpha}[x_i], y_i)$
13:        end if
14:    end for
15:    for each $i$ from 0 to $(L_{x2} - L_{x1})/H_G$ do
16:        for each $j$ from 0 to $(L_{y2} - L_{y1})/H_G$ do
17:            $x \leftarrow L_{x1} + H_G \cdot i$
18:            $y \leftarrow L_{y1} + H_G \cdot j$
19:            if $x \notin (L_x(y), U_x(y))$ or $y \notin (L_y(x), U_y(x))$ then
20:                Add grid nodes $\vec{a_{ij}} = (x, y, Z_\alpha)$
21:            end if
22:        end for
23:    end for
24: end procedure
```

$\max(x \mid (x, y, z) \in \mathcal{P})$, ϵ is a little value, and assign it label 1, resulting in the protein region being assigned label 2. Of course, multiple methods can be used in this step.

6 Membrane Extraction Algorithms

In our framework, we aim to partition the entire domain into three distinct regions: solvent, protein, and membrane, using a tetrahedral mesh. Currently, both the membrane and solvent regions are labeled as 1, so we still need to differentiate between these two. In other words, we need to distinguish between the 'pore' region of the protein and the outer solvent. This step is critical in our project, and there are various methods to achieve it. For instance, ICMPv2 [2] employs the Python package *trimmesh* to conduct a ray test on each tetrahedron to determine whether it is enclosed by the protein-membrane surface. Also, Liu's team uses the graph search algorithm based on straight walk on six directions. [5]

In the course of this project, we explored three efficient graph search algorithms. Given that the *neigh* file contains information about neighboring nodes, we can represent the tetrahedral meshes as a graph. In this graph, each tetrahedron is a vertex, and an edge exists between two vertices if their corresponding tetrahedra are neighbors. These relationships enable us to conduct a graph search. We will detail these three algorithms in the following sections of this report.

To begin, we label all tetrahedra within the solvent and membrane region (currently labeled as 1) that fall within the frame $[L_{x1}, L_{x2}] \times [L_{y1}, L_{y2}] \times [Z_1, Z_2]$ as membrane (designated by the label 3 in this project's framework). This approach inevitably results in over-labeling within the pore region. As a next step, we employ search algorithms to accurately identify the pore region and reassign these over-labeled tetrahedra as solvent.

6.1 Sub Meshes Partition

ICMPV2 [2] uses a sub-mesh partition to partition a large enough search area, therefore, reducing the number of tetrahedral we have to visit during the search process. Given a small value τ, and denoting the tetrahedral mesh of the entire box domain as D_h, we define a frame of the search area $\mathcal{S}$, and the sub-mesh $\mathcal{S}_h$ as follows:

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

$$\mathcal{S} = [a, b] \times [c, d] \times [Z_1, Z_2]$$
$$\mathcal{S}_h = \{\sigma \in D_h \mid \sigma \cap \mathcal{S} \neq \varnothing\}$$

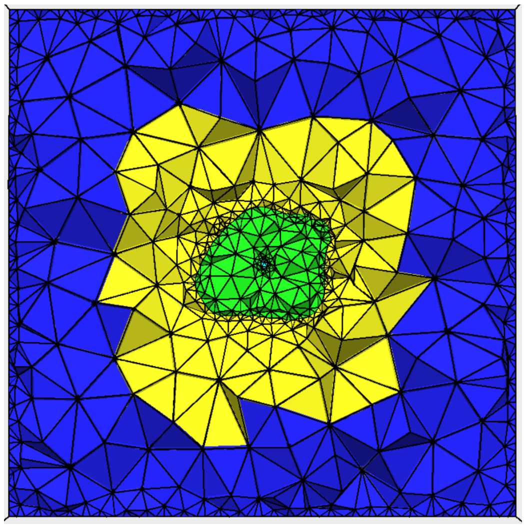


Figure 4: Membrane Search Partition Visualization

Figure 4 provides a visualization of the search region partition through a cross-section along the z-axis. The tetrahedral outside the search region are colored blue; those within the membrane portion of the search region are depicted in yellow; the protein is shown in green; and the pore regions, which are labeled as solvent, are represented in azure.

Since the search region enclosed the whole protein inside, we are confident that those not included by the search region are membrane tetrahedral and there is no meaning to look through them.

6.2 Algorithm - Breadth First Search

The Breadth-First Search (BFS) algorithm is a popular tool for graph search tasks, and we incorporated its central principles into this project. Given the well-structured nature of the ion channel protein, we can safely assume that within the search frame, there are two connected regions when considering only tetrahedral labeled as membrane (not protein). By conducting a BFS that begins from a randomly selected tetrahedron, we are able to differentiate between the membrane and solvent regions. However, as this initial tetrahedron may belong to either region, we must track the maximum x-value encountered during the search process. This value is then compared with P_x, the maximum x-value of the protein surface mesh obtained in earlier stages of the project.

Algorithm 2 Membrane Solvent Extraction (BFS)

```
1: procedure MEMBRANE SOLVENT EXTRACTION($T, V, N, D_{ms,h}, P_x$)
2:     $Q \leftarrow$ empty queue
3:     $M_x \leftarrow$ Lowest Number
4:     find a random tetrahedron $s \in D_{ms,h}$
5:     $mark[s] \leftarrow true$
6:     $Q.enqueue(s)$
7:     while Q is not empty do
8:         $v \leftarrow Q.dequeue()$
9:         for each neighbor $u$ of $v$ in $N(v)$ do
10:            if $mark[u] = false$ and $u \in D_{ms,h}$ and LABEL($u$) is not protein then
11:                $M_x \leftarrow max(M_x, \text{GETX}(u))$
12:                $mark[u] \leftarrow true$
13:                $Q.enqueue(u)$
14:            end if
15:        end for
16:    end while
17:    for each tetrahedron $t$ in $D_{ms,h}$ do
18:        if $M_x > P_x$ then
19:            if $mark[t] = false$ then label $t$ as solvent
20:            end if
21:        else
22:            if $mark[t] = true$ then label $t$ as solvent
23:            end if
24:        end if
25:    end for
26: end procedure
```

6.3 Algorithm - Two step Breadth First Search

Ion channels are proteins with distinct structures. Often, the solvent region within the pore region of the protein is significantly smaller than the protein's overall size. Therefore, during our BFS process, initiating the search from a tetrahedron within the pore region results in fewer tetrahedrons to visit, thereby improving our search efficiency.

In our two-step BFS approach, we first execute the BFS on one cross-section parallel to the Z-plane within the search frame. This initial step assists us in locating a tetrahedron within the solvent region. Then, in the second BFS step, we commence our search from the located solvent tetrahedron. As we traverse through the graph, we adjust the labels of the visited tetrahedrons accordingly.

Algorithm 3 Membrane Solvent Extraction (Two Step BFS)

```
procedure MEMBRANE SOLVENT EXTRACTION($T, V, N, D_{ms,h}, D^c_{ms,h}, P_x$)
    $Q \leftarrow$ empty queue
    $M_x \leftarrow$ Lowest Number
    find a random tetrahedral $s \in D^c_{ms,h}$
    $mark[s] \leftarrow true$
    $Q.enqueue(s)$
    while Q is not empty do
        $v \leftarrow Q.dequeue()$
        for each neighbor $u$ of $v$ in $N(v)$ do
            if $mark[u] = false$ and $u \in D^c_{ms,h}$ then
                $M_x \leftarrow max(M_x, \text{GETX}(u))$
                $mark[u] \leftarrow true$
                $s \leftarrow u$
                $Q.enqueue(u)$
            end if
        end for
    end while
    if $M_x > P_x$ then
        for each tetrahedron $t$ in $D^c_{ms,h}$ do
            if $mark[t] = false$ then
                $s \leftarrow t$
                break
            end if
        end for
    end if
    $Q.enqueue(s)$
    while Q is not empty do
        $v \leftarrow Q.dequeue()$
        label $v$ as solvent
        for each neighbor $u$ of $v$ in $N(v)$ do
            if $mark[u] = false$ and $u \in D_{ms,h}$ then
                $mark[u] \leftarrow true$
                $Q.enqueue(u)$
            end if
        end for
    end while
end procedure
```

6.4 Algorithm - Utilizing the Straight Walk to Identify the Cross Section

In the two-step BFS process, identifying the cross section could potentially be a time-consuming task. This is particularly the case if we continuously search through the lists of tetrahedrons and nodes to determine if a tetrahedron intersects with a plane defined by $z = C$.

An alternative approach is to use the straight walk algorithm, which is designed to locate all tetrahedrons that would intersect with a straight line drawn from point q to point p [1]. Leveraging this algorithm allows us to start from an arbitrary tetrahedron and move along the x or y axis, thereby constructing a contiguous submesh of the cross section.

With this submesh, we can then execute the BFS algorithm, leading to a more efficient identification of the cross section compared to the traditional method.

Algorithm 4 Straight Walk

```
procedure STRAIGHT WALK($u, v, w, t, p, q$)
    Initialize $T \leftarrow$ empty array
    while orientation($u, w, v, p$) $> 0$ do
        append $t$ to $T$
        $t \leftarrow$ neighbor($t$ through $uvw$)
        $s \leftarrow s \in V(t) - \{u, v, w\}$
        if orientation($u, s, q, p$) $> 0$ then
            if orientation($v, s, q, p$) $> 0$ then
                $u \leftarrow s$
            else
                $w \leftarrow s$
            end if
        else
            if orientation($w, s, q, p$) $> 0$ then
                $v \leftarrow s$
            else
                $u \leftarrow s$
            end if
        end if
    end while
    Output $T$
end procedure
```

Algorithm 5 Use Straight Walk to Find Cross Section

```
procedure FIND CROSS SECTION($T, V, N, D_{ms,h},$)
    $Q \leftarrow$ empty queue
    $D^c_{ms,h} \leftarrow$ empty array of tetrahedra
    find a random tetrahedral $s \in D_{ms,h}$
    $mark[s] \leftarrow true$
    $Q.enqueue(s)$
    while Q is not empty do
        $v \leftarrow Q.dequeue()$
        $v_1 \leftarrow v.Node1$
        $N(v) = \{(v_1.x - h, v_1.y, v_1.z), (v_1.x + h, v_1.y, v_1.z), (v_1.x, v_1.y - h, v_1.z)$
    $, (v_1.x, v_1.y + h, v_1.z), \}$
        for each neighbor $u$ in $N(v)$ do
            $t \leftarrow$ STRAIGHT WALK($v_1, u$)
            $t' \leftarrow$ last element of $t$
            $mark[t] \leftarrow true$
            Append $t$ into $D^c_{ms,h}$
        end for
    end while
    Output $D^c_{ms,h}$
end procedure
```

7 Numertical Results

We have run our program *MeshGen* on several ion channel proteins. In this section, we will present the protein surface mesh, the cross-section, and two three-dimensional views of the tetrahedral mesh that we have generated, respectively, for each ion channel protein.

7.1 Tetrahedral Meshes Generation Example Figures

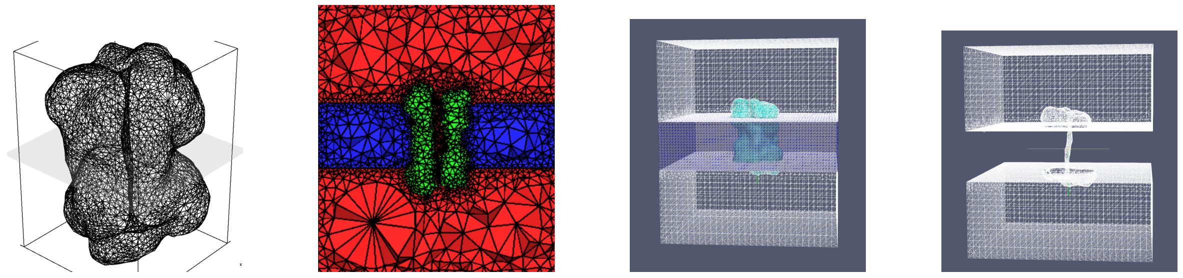


Figure 5: Protein - 1mag, generated by *MeshGen* with parameter: $Z_1 = -11 \quad Z_2 = 6 \quad tms_d = 0.5$

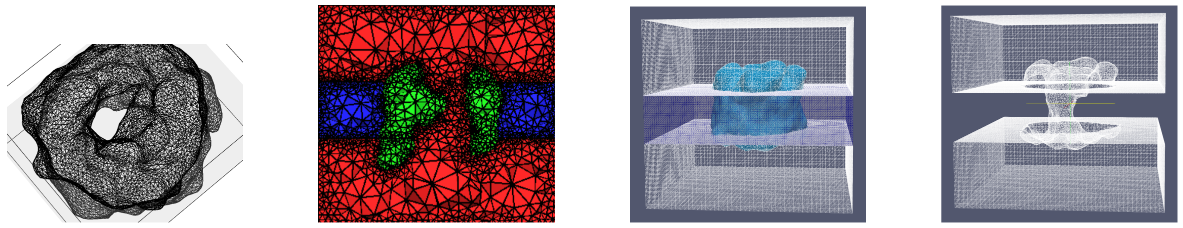


Figure 6: Protein - 3emn, generated by *MeshGen* with parameter: $Z_1 = -11 \quad Z_2 = 11 \quad tms_d = 0.2 \quad tetgen_q = 1.2$

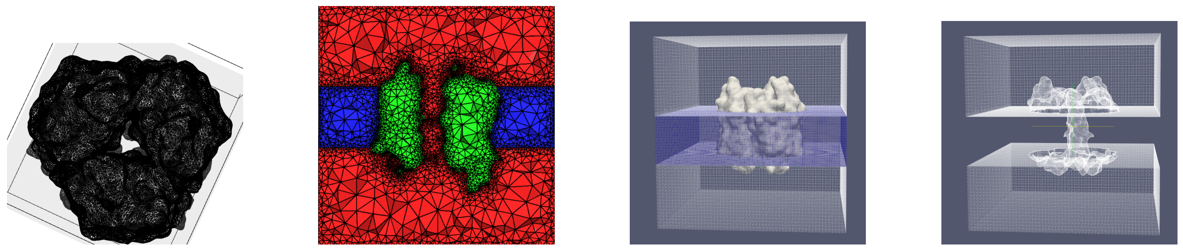


Figure 7: Protein - 3emn, generated by *MeshGen* with parameter: $Z_1 = -15 \quad Z_2 = 15 \quad tms_d = 0.3 \quad \eta = 25 \quad tetgen_q = 1.2$

7.2 CPU Runtime and Stats

During this project, key performance metrics, such as CPU runtime for each step in the program, the number of generated tetrahedra and vertices (or nodes) in the resulting mesh, as well as the count of tetrahedra involved in the search area during the membrane extraction process, were meticulously recorded.

Protein name	1mag	1ap9	3emn
TMSmesh	9.389	13.671	1.293
Box Surface Mesh	0.011	0.036	0.020
Add Membrane Surface vtx	0.009	0.071	0.023
Write Poly File	0.001	0.003	0.001
Mesh Generation (Tetgen)	7.303	42.548	19.126
BFS Search	0.520	4.075	1.652
2-Step BFS Search	0.490	3.684	1.470
Write XML File	0.208	1.264	0.54672
# Tetrahedral	303925	1746826	765165
# Vtxs	54991	302186	136900
# Tetrahedral in Search	39330	279128	99557

8 Disscusion during this project

8.1 Use of Dynamic Memory Locating

This project is highly memory-intensive. Even the simple task of reading the results from the *TMSMesh* software requires substantial memory. During our work on the program, we discovered that dynamic memory allocation for vertices and complex data is necessary in most instances to prevent segmentation faults. However, we believe that through careful implementation of dynamic memory management techniques, we can effectively regulate our memory usage and prevent any related issues.

8.2 Irregular Surface Mesh Tetrahedralization

The q switch parameter for *Tetgen* [7] governs the maximum radius-edge ratio. We need this value to be slightly larger than the separation distance of the membrane surface mesh points, S_b, that we integrated into the surface mesh. This is due to our desire for all membrane tetrahedra;l on the surface to include membrane surface mesh points, as well as to ensure that no tetrahedral penetrates the plane $z = Z_1$ or $z = Z_2$. If we set the q switch too small, the *Tetgen* software might insert additional vertices near the surface membrane mesh points, leading to irregular tetrahedral close to the membrane surface.

9 Appendix

The main source code for the mesh generation software introduced in this project, *MeshGen* can be found on the following github page: `https://github.com/dannydingx/mesh_generation`.

10 Acknowledgement

I am very grateful for my mentor, Zhen Chao, who is also the author of the ICMPV2 paper [2] that we referred to in this research. His expertise and insights were instrumental in shaping this study. He provided a deep explanation of his previous paper and helped me to understand his purpose in applying different methods. Thanks to his guidance, I found the content of computational biology to be very appealing.

References

[1] *Walking in a triangulation.* Association for Computing Machinery, 2001.

[2] Zhen Chao, Sheng Gui, Benzhuo Lu, and Dexuan Xie. Efficient generation of membrane and solvent tetrahedral meshes for ion channel finite element calculation, 01 2022.

[3] Zhen Chao and Dexuan 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*, 42:1929–1943, 08 2021.

[4] Minxin Chen and Benzhuo Lu. Tmsmesh: A robust method for molecular surface mesh generation using a trace technique. *Journal of Chemical Theory and Computation*, 7:203–212, 11 2010.

[5] Tiantian Liu, Shiyang Bai, Bin Tu, Minxin Chen, and Benzhuo Lu. Membrane-channel protein system mesh construction for finite element simulations. *Computational and Mathematical Biophysics*, 3, 11 2015.

[6] Christopher Maffeo, Swati Bhattacharya, Jejoong Yoo, David Wells, and Aleksei Aksimentiev. Modeling and simulation of ion channels. *Chemical Reviews*, 112(12):6250–6284, 2012. PMID: 23035940.

[7] Hang Si. Tetgen a quality tetrahedral mesh generator and 3d delaunay triangulator version 1.6 user's manual, 08 2020.