Commun. Comput. Phys.
doi: 10.4208/cicp.161114.021015a

Vol. **19**, No. 3, pp. 582-602
March 2016

Automated Parallel and Body-Fitted Mesh Generation in Finite Element Simulation of Macromolecular Systems

Yan Xie[1], Tiantian Liu[1], Bin Tu[2], Benzhuo Lu[1,*] and Linbo Zhang[1,*]

[1] *State Key Laboratory of Scientific and Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering Computing, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China.*
[2] *National Center for NanoScience and Technology, Chinese Academy of Sciences, Beijing 100190, China.*

Received 16 November 2014; Accepted (in revised version) 2 October 2015

Abstract. Mesh generation is a bottleneck for finite element simulations of biomolecules. A robust and efficient approach, based on the immersed boundary method proposed in [8], has been developed and implemented to generate large-scale mesh body-fitted to molecular shape for general parallel finite element simulations. The molecular Gaussian surface is adopted to represent the molecular surface, and is finally approximated by piecewise planes via the tool `phgSurfaceCut` in PHG [43], which is improved and can reliably handle complicated molecular surfaces, through mesh refinement steps. A coarse background mesh is imported first and then is distributed into each process using a mesh partitioning algorithm such as space filling curve [5] or METIS [22]. A bisection method is used for the mesh refinements according to the molecular PDB or PQR file which describes the biomolecular region. After mesh refinements, the mesh is optionally repartitioned and redistributed for load balancing. For finite element simulations, the modification of region mark and boundary types is done in parallel. Our parallel mesh generation method has been successfully applied to a sphere cavity model, a DNA fragment, a gramicidin A channel and a huge Dengue virus system. The results of numerical experiments show good parallel efficiency. Computations of electrostatic potential and solvation energy also validate the method. Moreover, the meshing process and adaptive finite element computation can be integrated as one PHG project to avoid the mesh importing and exporting costs, and improve the convenience of application as well.

AMS subject classifications: 65N50, 65Y05, 92C40, 65N30

Key words: Parallel mesh generation, body-fitted, PHG, biomolecules, finite element simulation.

*Corresponding author. *Email addresses:* `xieyan@lsec.cc.ac.cn` (Y. Xie), `liutt@lsec.cc.ac.cn` (T. Liu), `tubin@lsec.cc.ac.cn` (B. Tu), `bzlu@lsec.cc.ac.cn` (B. Lu), `zlb@lsec.cc.ac.cn` (L. Zhang)

1 Introduction

Continuum model uses a continuum description of the discrete particles (e.g., water molecules, ions, or protein molecule), which has been widely used in molecular simulations as an effective way to reduce the computational cost. The Poisson-Boltzmann equation (PBE) represents a typical continuum model describing the electrostatic interactions and ionic density distributions of a solvated molecular system at equilibrium state. The most commonly and practically used numerical methods to solve the PBE include finite difference (FD) method [2,33], finite element method (FEM) [20,29,35,37,39], and boundary element method (BEM) [4,25,27,42].

So far, mesh generation is still a bottleneck for usage of FEM/BEM due to the highly irregular shape of biomolecular systems. For biomolecular simulations, this task is further complicated by the identification of the irregular molecular surface and an appropriate description of this surface for resolving the molecular structures in sufficient details. For boundary element method a molecular surface mesh suffices; while for finite element method or its coupling with boundary element methods, a volume mesh in the solvent region and/or the solute region is also needed [28]. Triangular and tetrahedral meshing are most widely used forms of unstructured mesh generation [32]. There have been a number of free programs developed to generate surface triangular meshes for biomolecules. However, few free software can be used to generate tetrahedral meshes for biomolecules directly. A frequently-used package, Tetgen [36], is able to generate tetrahedral meshes based on the surface triangular meshes. A common strategy to obtain the tetrahedral meshes for biomolecules is generating the surface triangular meshes first and then obtaining the tetrahedral volume meshes based on the surface meshes.

We have built a tool chain to generate high-quality meshes for practical protein systems by combining a few mesh generation tools which are based on Delaunay meshing. The tool chain has essentially these components: surface meshing, quality improving, volume mesh generation, and membrane-protein mesh construction is necessary. First, a triangulation of the Gaussian surface is generated using our recently developed program TMSmesh [6], which is a robust tool for meshing molecular Gaussian surfaces and has been shown to be capable of handling molecules consisting of more than one million atoms. In the second step, the program ISO2Mesh [17] is first used to simplify the surface mesh by reducing the number of faces or adding some nodes while preserving its manifoldness, volume, and boundary shape. If self-intersecting faces exist, then the program TransforMesh [41], which can robustly handle topology changes and remove self-intersections, is used to find and remove self-intersecting faces. Finally, in the third step, a tetrahedral volume mesh is generated using the program TetGen, which consists of four-node tetrahedral elements and is ready for 3D finite element simulations. More details can be found in [7]. With this tool chain we have successfully generated meshes for many protein systems and performed finite element simulations on them [37,40].

Today's parallel computers enable us to solve a problem with a mesh containing tens of millions of vertices. However, CPU time and memory limitations still make it a challenging task to generate such a large high-quality mesh on a single machine. Obviously, a parallel environment significantly reduces the amount of time required for large scale mesh generation [21]. Although there is no widely-used free parallel mesh generator by now, parallel techniques for distributed-memory machine are described in many research papers focusing on Advancing Front methods [13,26,31], Delaunay methods [11,34] and Octree-based methods [14] respectively. In order to generate high-quality meshes for finite element simulations, a powerful mesh generator should satisfy the following requirements:

1. High parallel efficiency for meshing;
2. Easy implementation;
3. The meshing process and finite element computation can be integrated in the same program to avoid the mesh importing and exporting costs.

In this paper, we propose a method to generate in parallel tetrahedral meshes directly from the initial molecules instead of from existing surface triangular meshes. Our code is based on the parallel adaptive finite element package PHG [43], which is written in C and uses MPI for message passing. PHG provides many interfaces for computation and meshing and hides the parallelization details for easy implementation with thousands of CPU cores. Based on the adaptive mesh refinement interfaces in PHG, we successfully develop and implement parallel unstructured mesh generation algorithms for a given molecular PQR file.

In the following section, we elucidate the framework of mesh generation and describe the details of parallelization and implementation techniques. After the whole meshing process, the final mesh is exported to a file in the Medit [18] file format. Some protein systems and ion channels are taken as examples in Section 3, and the parallel efficiency and NPBE simulations are given in the subsections. Final conclusions are given in Section 4.

2 Automated parallel and body-fitted mesh generation in finite element simulation of macromolecular systems

2.1 Region for meshing

Solvated biomolecular systems are usually modeled by dielectric distinguished regions with singular charges distributed in the molecular region. Systems without singular charges or dielectric jump are usually found in simplified models with planar or cylindrical boundary geometries in electrochemistry and biopolymer science, and can be regarded as a special case of the systems in this investigation. Fig. 1 schematically illustrates a solvated biomolecular system occupying a domain Ω with a smooth boundary $\partial\Omega$. The solute (molecule) region is represented by Ω_m and the solvent region by Ω_s. The dielectric interface Γ_m is defined by the molecular surface. The process of generating

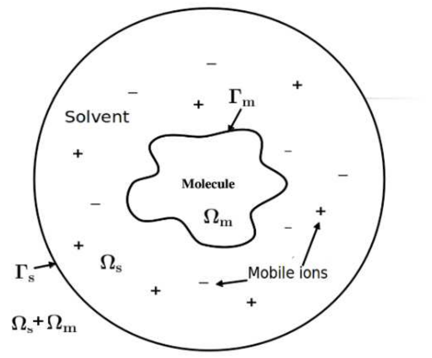


Figure 1: Two-dimensional illustration of the region for meshing.

tetrahedral meshes is done in all regions as showed in Fig. 1. For finite element computation, the generated mesh should capture important geometric characteristics and correctly mark regions and boundary types.

2.2 Parallel unstructured mesh generation

Our mesh generation scheme follows these steps: 1) Import a background mesh, partition it and distribute it to processes. Read PQR file (see Appendix A) meanwhile; 2) refine the mesh and mark the regions by applying adaptive refinement interfaces of PHG; 3) add Gaussian surface into the mesh; 4) check and modify region marks to guarantee nonexistence of incorrect marks; 5) mark inner and outer surfaces and export the mesh in Medit format. Step 3 is optional which can be skipped as users prefer. The framework is illustrated in Fig. 2.

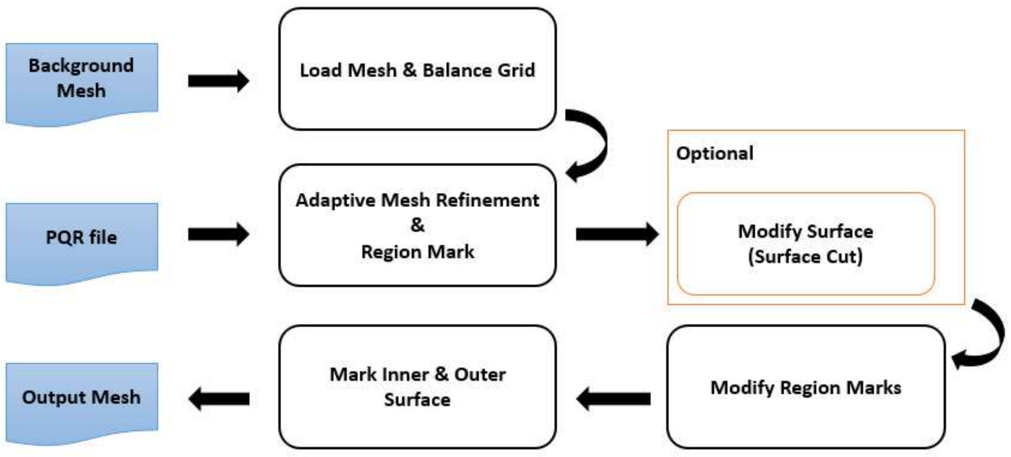

Background Mesh
Load Mesh & Balance Grid
PQR file
Adaptive Mesh Refinement & Region Mark
Optional
Modify Surface (Surface Cut)
Output Mesh
Mark Inner & Outer Surface
Modify Region Marks

Figure 2: The framework of parallel adaptive mesh generation.

2.2.1 Load and partition the mesh

Firstly, we load a mesh file and partition the mesh for parallel processing and load balancing. In order to distribute a mesh $\mathcal{T}$ on a distributed memory computer, it is partitioned into P submeshes $\mathcal{T}_i$, $i=0,\cdots,P-1$, where P is the number of MPI processes. The partitioning is computed using either a PHG's built-in mesh partitioning method such as Hilbert space filling curve [5], or existing tools like METIS [22]. The partitioning is element-based, i.e., the set of tetrahedra of $\mathcal{T}$ is divided into P disjoint subsets and each subset is assigned to an MPI process. Every time after mesh refinement, if the *load imbalance factor* (LIF, the maximum number of tetrahedra in a submesh over the average number of tetrahedra in the submeshes) exceeds a given threshold, a new mesh partitioning is computed and the submeshes are redistributed to maintain load balance.

2.2.2 Mark the regions

The region mark of each element in the mesh is used to distinguish the solute and solvent regions. All elements intersecting with the molecule surface are regarded as in the solute region, while others are in the solvent region. An intuitive strategy to get the region mark is that each process checks the intersections of elements with the molecular surface by traversing the atoms in the PQR file. The region mark is set to 1 if the element intersects with the molecular surface and 0 otherwise. The time complexity of this method is $\mathcal{O}(\frac{NM}{n})$, where N denotes the number of atoms of the molecule, n the number of processes, M the number of tetrahedra in the submesh. A molecular volume mesh sometimes contains millions or even more tetrahedra and updating the region mark can cost minutes or even hours, which is unacceptable.

The process of computing the region marks is accelerated by dividing the atoms into a given number of buckets. Then only atoms in a limited number of buckets instead of all atoms need to be traversed for each element, using a look-up table. The pseudo codes are given in Algorithm 1. The time complexity of Algorithm 1 is $\mathcal{O}(\frac{NM}{nk})$, where k denotes the number of the buckets. In our implementation, process 0 creates the look-up table and broadcasts it to all processes. Suitable values for dx, dy and dz are determined such that the number of buckets is around 10^4-10^5. Timing results with Algorithm 1 will be shown in Section 3.3.

2.2.3 Refine the mesh

In this step, our target is to refine the mesh to get a discretized molecular representation with a high resolution. In practice, only the elements near the interface, i.e., those whose region mark is different from one of its neighbours, are refined for saving memory. Elements are refined using the bisection method as illustrated in Fig. 3. The parallel mesh refinement is done in two phases. In the first phase, the submeshes are refined independently, with the shared faces of any two submeshes treated as if they were boundaries. This step creates locally conforming submeshes, but nonconforming across submeshes. In the second phase, nonconforming faces between submeshes are exchanged and ele-

Algorithm 1 Create a look-up table for atoms of a molecule.

```
Read the PQR file and update coordinates ranges (int)$x$_min, $x$_max, $y$_min, $y$_max,
$z$_min and $z$_max;
dx, dy, dz denote the length in each direction of the bucket box, $\mathrm{kx}=\frac{x\_max-x\_min}{dx}+1$,
$\mathrm{ky}=\frac{y\_max-y\_min}{dy}+1$ and $\mathrm{kz}=\frac{z\_max-z\_min}{dz}+1$ denote the number of the buckets in each
direction;
Define map<int, vector<int> > Buckets, the key of map denotes the index of the
bucket, the value of map denotes the PQR atoms in the specific bucket.
for $i=0$ to $N-1$ do
    $\mathrm{index}=\mathrm{ky}*\mathrm{kz}*\frac{pqr[i].x-x\_min}{\mathrm{dx}}+\mathrm{kz}*\frac{pqr[i].y-y\_min}{\mathrm{dy}}+\frac{pqr[i].z-z\_min}{\mathrm{dz}}$;
    if Buckets.find(index) != Buckets.end() then
        Buckets[index].push_back(i);
    else
        vector<int> tmp;
        tmp.push_back(index);
        Buckets[index] = tmp;
    end if
end for
```

Algorithm 2 Mesh Generation Based on PHG Adaptive Refinement Interfaces.

```
Load an initial background mesh $\mathcal{M}_0$;
Set the maximum iteration $N$;
$k=0$;
while $k<N$ do
    Update the region mark of each element in $\mathcal{M}_k$;
    Refine the element according to the marks and keep conformity. $\mathcal{M}_k$ denotes the
    new mesh;
    Check load balancing and repartition the mesh if necessary.
    $k=k+1$;
end while
```

ments having inter-submesh nonconforming faces are refined. The process is repeated until the global conformity of the mesh is reached. Algorithm 2 shows the parallel mesh refinement process. This algorithm is easy to implement using PHG's adaptive mesh refinement interface. A sample mesh is shown in Fig. 4 for a sphere cavity model with a radius of 200$\AA$ and a solvent region of a unit sphere. The background mesh, which contains 498 vertices and 1810 tetrahedra, is generated by TetGen. Fig. 4(a,b) shows the surface changes while the mesh is repeatedly refined using Algorithm 2. Obviously, the solute surface fits the curve better but is bumpy even if the mesh gets very fine. A validation of the mesh generation method is demonstrated by solving the Nonlinear Poisson-

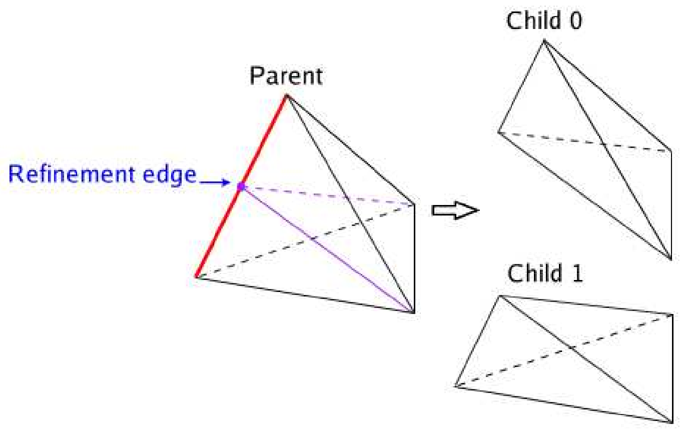


Figure 3: Bisection of a tetrahedron.

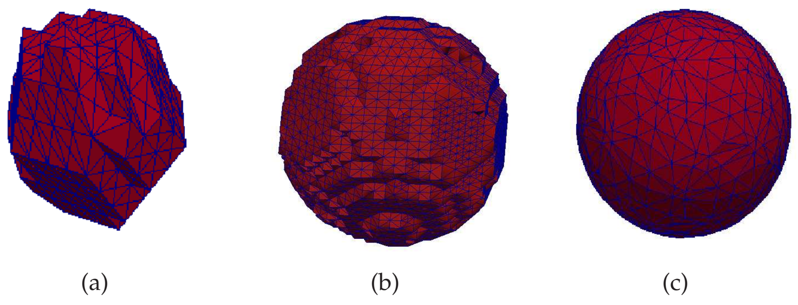


Figure 4: The surface of the solute region for the sphere cavity model. (a) 20 refinements; (b) 30 refinements; (c) 20 refinements and use `phgSurfaceCut` to fit the analytical surface.

Boltzmann equation on the sphere cavity model. It is found that the electrostatic potential is radial symmetric and the maximum value is 3.78 kcal/mol$\cdot e$ locating near the surface, which is consistent with [29].

2.2.4 Fit the surface (surface cut)

After the above steps, the surface of the solute region is zigzag and bumpy, so we need to smooth the surface for fitting the actual interface and avoid possible numerical problems in finite element computation. Our method follows these steps: 1) Calculate all intersection points (cut points) between the analytical surface and edges in the mesh, since the surface is locally approximated by planes, at most one cut point is allowed on each edge. 2) Add the cut points to the mesh and subdivide elements containing cut points while maintaining the conformity of the mesh. 3) Approximate the analytical surface with cut planes. PHG provides a tool, `phgSurfaceCut`, to create a new conforming tetrahedral mesh with respect to a given set of cut points by locally subdividing all tetrahedra containing cut points into smaller tetrahedra. That tool was used in [8] and it worked well for the test cases with simple surfaces presented in [8], but is not robust enough and may crash for complicated surfaces as those addressed in this paper. Thus we have revised and improved the underlying algorithm used in the `phgSurfaceCut` function and it's now robust and can reliably generate a new mesh with any given surface. Below we give a brief description of the new algorithm.

Our new algorithm for computing the intersection of a surface with a mesh is divided into two steps. The first step consists of computing the cut points and the second step consists of subdividing the tetrahedra with respect to the given cut points. To ensure robustness of the algorithm, the cut points generated in the first step must satisfy the constraints on the distribution of the cut points in each element required in the second step. We will first describe the second step, then the first step of the algorithm.

In the second step of the algorithm, a set of cut points is taken as input and elements containing cut points are subdivided to create a new conforming mesh. The cut points are given on the edges. Since in each element the surface is locally approximated by a plane, at most one cut point is allowed on each edge, and in each element only the following cases are considered:

1. There is no cut point on the element. The element is unchanged.
2. There is 1 cut point on the element. The element is bisected into 2 tetrahedra (Fig. 5(a)).
3. There are 2 cut points on the element. The element is subdivided into 3 tetrahedra by first subdividing it into a tetrahedron and a pyramid and then subdividing the pyramid into 2 tetrahedra (Fig. 5(b)).
4. There are 3 cut points on the element, in this case the 3 edges containing the cut points must have a common vertex. The element is subdivided into 4 tetrahedra by first subdividing it into a tetrahedron and a prism and then subdividing the prism into 3 tetrahedra (Fig. 5(c)).
5. There are 4 cut points on the element, in this case the four cut points must lie on two pairs of opposite edges. The element is subdivided into 6 tetrahedra by first subdividing it into two prisms and then subdividing each prism into 3 tetrahedra (Fig. 5(d)).

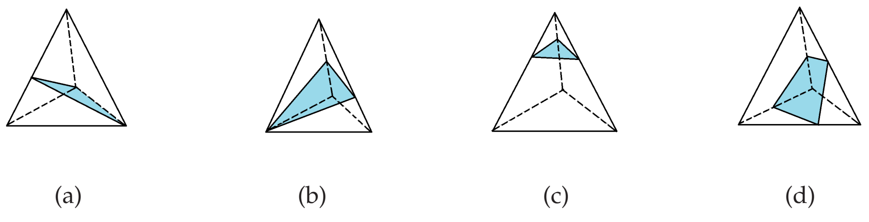

(a) (b) (c) (d)

Figure 5: Intersections between a plane and a tetrahedron.

All other distributions of cut points on an element are not allowed.

Now we describe the first step of our algorithm, which is responsible for generating a set of cut points satisfying the requirements above. In this step, a user function defined in the domain is taken as input which defines the surface, it has positive values on points at one side of the surface, negative values on points at the other side of the surface, and zero on the surface. A set of cut points is generated using the user function as follows:

1. A 'signness' of -1, $+1$, or 0 is assigned to each vertex of the mesh depending on whether the value of the user function on it is negative, positive or zero.

2. On each edge whose two vertices have different and nonzero signness, a cut point is computed using the user function with either bisection or linear interpolation (in the latter case the user function is assumed continuous).

3. If a cut point is closer to a vertex than a given threshold, then the signness of the vertex is changed to 0 and all the cut points on the edges connected to the vertex are removed. This is to avoid very close vertices in the new mesh.

It can be shown that the set of cut points thus generated fully satisfies the requirements in the second step of the algorithm, thus the robustness of the whole algorithm is ensured.

For the sphere cavity model, the solute surface has an analytical formulation and the user function can be analytically given using the distance between the center and the vertices. Fig. 4(c) illustrates the solute surface generated by using `phgSurfaceCut` after 20 refinements. For practical molecules, the molecular surface may be defined in various ways [12, 19]. The most widely used molecular surfaces are 1) the van de Waals surface, which is the smallest envelope enclosing a collection of spheres representing all the atoms in the system with their van de Waals radii, 2) the solvent accessible surface (SAS), which is the trace of the centers of probe spheres rolling over the van de Waals surface, 3) the solvent excluded surface (SES), which is the surface traced by the inward-facing surface of the probe sphere, 4) Skin surface [9, 16, 24], which is defined by a set of weighted points representing the atoms and a scalar called the shrink factor controlling hyperboloidal connections between neighboring spheres, 5) Gaussian surface [38, 44], which is a level-set of the summation of the spherically symmetrical Gaussian density distributions centered at each atom of the biomolecular system, 6) the boundary of a domain enclosing the molecule such that certain energy is minimized over this boundary, such as the surface free energy [3]. In this paper, we use the Gaussian surface to generate molecular surface, which is given by: $\sum_{i=1}^{N} e^{-d(\|x-x_i\|^2-r_i^2)}-1=0$. We use the Gaussian surface as its real protein surface and approximate it with piecewise planes with `phgSurfaceCut`.

2.2.5 Modify region marks

In the above steps, it is difficult to avoid producing wrong region marks. More precisely, a correct mesh in our case should guarantee that there exists only one connected solute region (if the solute has no separate parts) and one solvent region respectively. However, many isolated regions may occur after region marking. To find out all isolated regions, we modify the region marks according to the following steps:

- **Step 1:** Mark all connected elements in each process and assign specific tags.
  In each process, find out all connected elements with a same region mark and generate tags for them. Connected elements share a common tag. The tags represent the region mark, the process rank and the ID. Taking a tag '1_2_1' for example, it tells that all elements with this tag are marked as the solute region and they are from process 2 and the ID is 1. After tagging, we merge the tags around the submesh boundaries as tag pairs and send them to the process 0.

- **Step 2:** Process 0 gathers all tag pairs on the submesh boundaries and allocates representative tags. The process 0 gathers lots of tag pairs. A tag pair, like (a,b), denotes that the region tagged as a is connected with the region tagged as b. A tag pair has different tags only because the elements are owned by different submeshes. Therefore we should allocate representative tags for all connected regions and then broadcast these mapping between initial tags and representative tags to all. This process can be modeled as finding connected subgraphs. Each node denotes a specific tag and each tag pair represents an edge. After the graph is built, we use Breadth First Search (BFS) method to find all connected subgraphs. For each subgraph, we randomly choose one tag as a representative tag and all tags in this subgraph are mapped to it. Process 0 then broadcasts the mappings to all processes.

- **Step 3:** Each process receives the representative tags from process 0 and modifies wrong region marks.
  After Step 2, all representative tags and mapping relations are broadcasted to all processes. Then we find out the top 2 tags with the most number of elements. In this paper, only two regions are taken into consideration, thus the top 2 tags are the solvent and solute regions respectively. Meanwhile, the other tags are isolated and regarded as wrong region marks, and we change the region mark as 1 if the initial value is 0, and vice versa.

2.2.6 Mark inner and outer surfaces

After the regions are correctly marked, we mark the outer surfaces as DIRICHLET and the inner surfaces as BDRY_USER1. Finally, the mesh is exported in Medit format. It's worth noting that the output mesh file is in text (not binary) format so that we use the root process to receive the data from each process and write them into the output file separately rather than using `MPI I/O` for parallel I/O. Actually, our codes of finite element computation are integrated with the meshing process, therefore the mesh exporting can be omitted in FEM modeling.

3 Numerical experiments and analysis

The implementation of the algorithms is based on the parallel adaptive finite element package PHG. The computations were carried out on the cluster LSSC-III of the State Key Laboratory of Scientific and Engineering Computing of China, which consists of computing nodes with dual Intel Xeon X5550 quad-core CPUs, interconnected via DDR InfiniBand network. In this section, we give some examples to describe the applications and performance of our mesh generation method.

3.1 Mesh generation for a DNA fragment

One of the protein systems introduced here is the DNA fragment embedded in a spherical computational domain with a radius of $200\AA$. The molecular surface is schematically illustrated in Fig. 6. This system carries out a total fixed charge of -22 e. The mesh over the whole domain has a total of 99093 vertices and 620117 simplices. The molecular surface contains 24503 vertices and 49002 triangles.

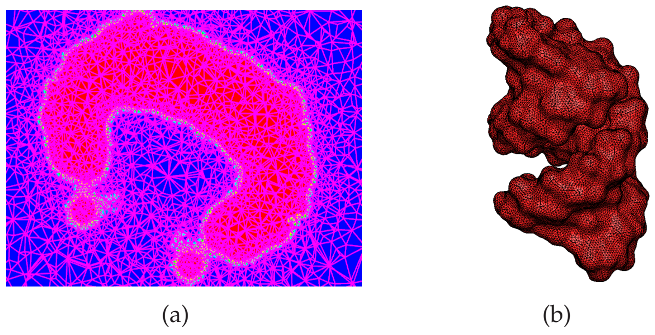


Figure 6: An example of mesh generation for a fragment of A-form DNA. (a) A close-up view of the fine mesh around the molecule. (b) The triangular boundary mesh conforming to the molecular surface.

3.1.1 Generated tetrahedral meshes

The meshes in Fig. 7 start from a background mesh which contains 129 vertices and 342 tetrahedra and are obtained after completing 6, 12 and 18 mesh refinements respectively after 8 uniform refinements using Algorithm 2. Different from the sphere cavity model, the DNA fragment does not have an analytical molecular surface, therefore we use the Gaussian surface to approximate it. Figs. 8(a) and 8(b) show the molecular surfaces by adding `phgSurfaceCut` after 12 and 18 refinements respectively. The surface meshes showed in Fig. 8 are smoother than the ones in Fig. 7.

3.1.2 Application to the PBE

In order to facilitate understanding of the results, we give a brief introduction to PBE. The nonlinear Poisson-Boltzmann equation (NPBE) reads:

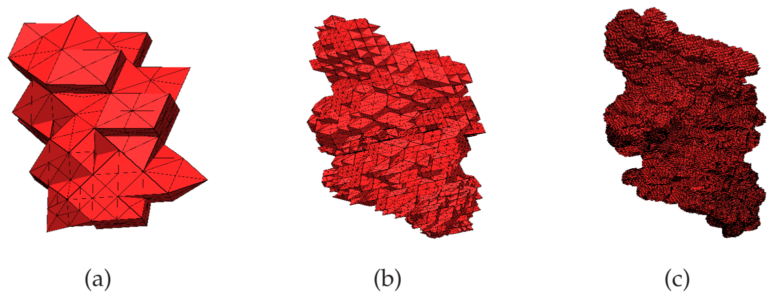


Figure 7: The solute surfaces of the DNA fragment after (a) 6 refinements; (b) 12 refinements; (c) 18 refinements.

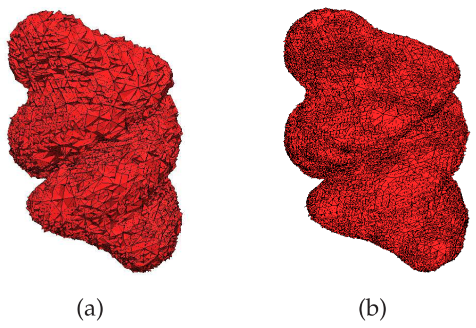


Figure 8: Approximate the solute surface by the Gaussian surface via `phgSurfaceCut`. (a) Add the Gaussian surface in the mesh of Fig. 7(b). (b) Add the Gaussian surface in the mesh of Fig. 7(c).

$$-\nabla\cdot\epsilon\nabla\phi+\kappa^2\sinh(\phi)=\rho^f, \tag{3.1}$$

where ϵ is a spatial-dependent dielectric coefficient, κ absorbs all the related parameters, and ρ^f and ϕ are the scaled singular charged distribution and electrostatic potential respectively. An effective strategy for solving the NPBE is to decompose ϕ into three parts, the singular component G, a harmonic component H and the regular component u_r as [10,30]

$$\phi=G+H+u_r. \tag{3.2}$$

We use inexact Newton iterative method to solve the nonlinear problem and more details can be found in [28,40].

Solvation energy is a reasonable metric to validate the accuracy of the mesh generation method. The definition of the solvation energy is given as Eq. (3.3), where q_i denotes the i-th atom's charge in the PQR file, u_i and H_i denote the regular component and harmonic component values at the center of the i-th atom respectively.

$$E_{\text{solvation}}=\frac{1}{2}\sum_i q_i(u_i+H_i). \tag{3.3}$$

Table 1: Electrostatic potential range and solvation energies of the DNA fragment with different meshes.

Mesh	Electrostatic potential (kcal/mol$\cdot e$)	Solvation energy (kcal/mol)
Mesh A	[-9.99, 5.08]	-4.5396×10^3
Mesh B	[-7.29, 0.01]	-3.0991×10^3
Mesh in Fig. 6 (Baseline)	[-7.72, 0.63]	-3.5160×10^3

The electrostatic potential and solvation energy values calculated from the NPBE results are listed in Table 1. Mesh A represents the mesh obtained by 18 refinements while Mesh B represents the mesh using `phgSurfaceCut` based on Mesh A. To get high-quality mesh, the cut points are set to the centers of the cutting edges when using `phgSurfaceCut`. It is found that all the solvation energies and electrostatic potentials are comparable, but the results of the mesh with introducing the Gaussian surface are much closer to those from obtained with the baseline. That is to say, the shape of the interface affects both the extreme values and the statistical values such as solvation energies.

3.2 Add membrane for ion channel

The mesh generation algorithm can also be applied to ion channels. Ion channels are set on the cell membrane, therefore creating the membrane for the channels is indispensable in the process of simulations for ion channels. The mesh generation algorithm is applied to a Gramicidin A (gA) channel protein (PDB code: 1MAG), which forms aqueous pores in lipid bilayers that selectively pass monovalent cations [1,23]. gA is a small 15-amino-acid β helical peptide with a narrow pore. As it is relatively small and well characterized experimentally, a wide variety of theoretical models have been applied to the gA channel. The whole domain of the gA channel consists of the membrane-protein region, bulk region, and the channel region. The partial charges and atomic radii for each atom in the protein are obtained by using the PDB2PQR software [15]. The gA channel pore region is along the z direction. The box size is $30\AA\times 30\AA\times 45\AA$. The background mesh has a total of 773 vertices and 3264 tetrahedra. A close-up view of the gA channel after 8 refinements is shown in Fig. 9(a), and Fig. 9(b) and (c) show the $y-z$ plane and $x-y$ plane view respectively.

`phgSurfaceCut` is used to smooth the solute surface and also can be applied to embed a membrane slab for ion channels. Since the membrane slabs in the z direction are analytical determined within: $z=z_1$ and $z=z_2$. Our method is to use `phgSurfaceCut` to embed the two faces and mark all the elements between these two faces as membrane region excluding the channel region and protein region, which is shown in the Fig. 10. Obviously, the membrane surface obtained by this method is very smooth and the processing is quite efficient and trivial. As for modifying region marks, users can utilize recursive method via starting from one element of the membrane region and visiting the neighbors. Users can also add more limitations as stop rules to control the marking process. Once

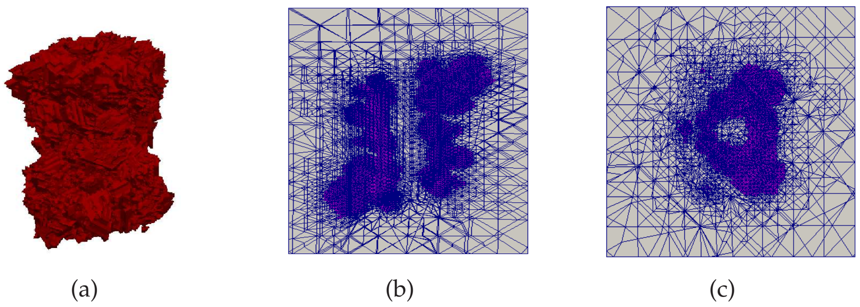


Figure 9: Mesh of the gA channel after 8 refinements based on 5 times uniform refinements. (a) The triangles of the solute surface; (b) $y-z$ plane at $x=0$; (c) $x-y$ plane at $z=-3.5$.

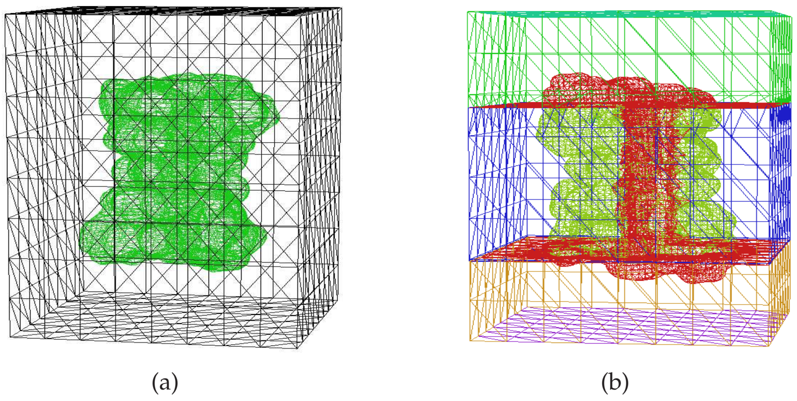


Figure 10: Embedding a membrane slab for the gA channel. (a) Gramicidin A channel; (b) Embedded membrane slabs and marked regions.

the region mark process ends, we should guarantee that the whole domain is composed of one membrane region, one protein region together with the membrane region and no isolated regions exist.

3.3 Mesh generation for the Dengue virus system

1K4R is the Dengue virus PDB code and the PQR file contains 1082160 atoms in the solute region. To generate a high resolution mesh for such a huge molecular system, users have to parallelize the generation process efficiently. 1K4R has a total of $-360e$ charges, the coordinates range (data type: `int`) is $[-251,251]\AA\times[-251,251]\AA\times[-251,251]\AA$. Because of memory limitation in Paraview, a mesh with more than 20 million vertices is hard to visualize. Fig. 11 illustrates a small mesh containing a total of 2015192 tetrahedra and 363561 vertices, and the molecular surface contains 553702 triangles. Fig. 12(a) is the molecular surface of the virus and it looks like a spherical envelope with pores scattered on it. Fig. 11 is a close-up view of the mesh, which shows the pores formed by the gaps among proteins.

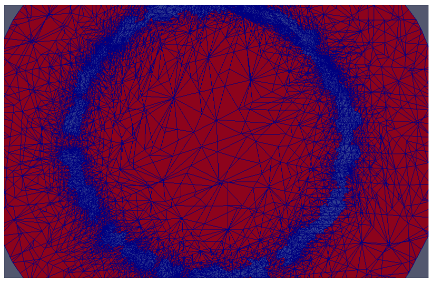
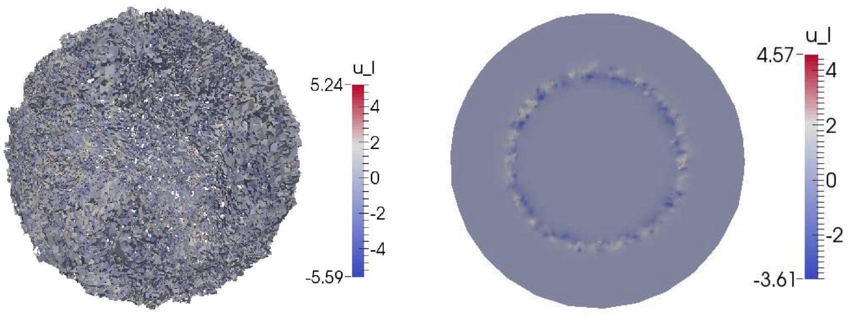


Figure 11: A close-up view of Dengue virus.



(a) (b)

Figure 12: Electrostatic potential distributions of 1K4R system. (a) Around the molecule; (b) at the $x-y$ plane through the center of domain.

As the numbers of atoms of 1K4R exceeds 1000000 and it costs several tens of minutes in each refinement if we use the first method in Section 2.2.2 to mark the elements. By contrast, Algorithm 1 is very robust and only costs seconds for each refinement. Table 2 elucidates the efficient refinement process, from which we can also observe the volume and area changes of the solute region during refinement. It's worth noting that the time complexity of `phgSurfaceCut` is $\mathcal{O}(\frac{MN}{k})$, where N denotes the number of atoms, M the number of edges and k the number of processes. Because the optimization of the user function for `phgSurfaceCut` is nontrivial, we skip the surface cut process for 1K4R case as its abundant atoms.

Using the generated mesh, we solve the nonlinear Poisson-Boltzmann equation in the same `main()` function and calculate the electrostatic potential distribution in the domain. The NaCl solution and the ionic strength is set as 50 mM. Fig. 12 shows the electrostatic potential around the molecule. The potential range is from -5.59 to 5.24 kcal/mol$\cdot e$ and is from -3.61 to 4.57 kcal/mol$\cdot e$ on the $x-y$ plane through the point $(0,0,0)$. From the figures, we can find that strong potentials are all distributed around the molecular surface and they debilitate in the solvent region and in the cavity. These simulation results are comparable to our numerical results obtained by Boundary Element Method with triangular meshes [42].

Table 2: Mesh refinements for a Dengue virus envelope (PDB code: 1K4R) based on Algorithm 1. 1K4R contains 1082160 atoms, and the coordinates ranges are x:[-251, 251]$\mathring{A}$, y:[-251, 251]$\mathring{A}$, z:[-251, 251]$\mathring{A}$. The following refinements use 32 processes.

Elements #	Volume($\mathring{A}^3$)	Area($\mathring{A}^2$)	CPU time(ms)
68827	3.0127e+07	3.0706e+06	1.604e+01
94134	2.7047e+07	3.0896e+06	2.798e+01
150092	2.4125e+07	3.1815e+06	5.500e+01
240304	2.1615e+07	3.2827e+06	1.007e+02
424142	1.9452e+07	3.4084e+06	1.998e+02
749231	1.7680e+07	3.6384e+06	3.761e+02
1307192	1.6210e+07	3.7901e+06	6.906e+02
2380102	1.5006e+07	3.9862e+06	1.326e+03
4249181	1.4069e+07	4.2311e+06	2.444e+03
7445395	1.3274e+07	4.4063e+06	4.576e+03
13404398	1.2584e+07	4.6462e+06	8.912e+03
23608643	1.1989e+07	4.9825e+06	1.701e+04

3.4 Parallel efficiency of mesh generation method

In our paper, all the unstructured meshes are generated on distributed-memory computer systems. To demonstrate the scalability, the same finer volume mesh is created using different numbers of processors. The original coarse volume mesh is partitioned and distributed into each processor. After mesh refinements, surface improvement (Surface Cut, optional), region mark modification and surface type marking, the fine volume mesh with specific boundary types is generated for finite element computation.

The numerical test is performed on the 1K4R case because of its large scale. It starts from a coarse background mesh with 773 vertices and 3264 tetrahedra. After many times refinements, the finer volume mesh has a total of 4207955 vertices and 23608643 tetrahedra. To validate the efficiency and scalability of our algorithm, we use different numbers of processors to get the volume mesh. The CPU times are shown in Table 3. Except for the cases in which the total number of processes is not a multiple of 8, 8 processes are used on each compute node so that we have exactly 1 process per core. Table 3 shows that our algorithm can generate a mesh containing more than 20 million tetrahedra within a minute by using 64 processors, which indicates a high efficiency of the algorithm.

Fig. 13 shows the speedup of the algorithm. Our test starts from 4 processes, thus it is the baseline of the speed-up. During the calculation, the CPU time consumption is mainly on the mesh refinement to generate a larger mesh. Fortunately, the parallel mesh refinement achieves good scalability, which reduces the computing time much. The speed up of mesh refinement is up to 24.8 when we use 64 processors. However, the performance of modifying the region marks becomes worse when the number of processors is increased because only the master processor handles the surface tags and the tags number increases

Table 3: The CPU times using different numbers of processors.

Processors	CPU Time (s)		
	Mesh Refinement	Modify Region Marks	Overall
4	116.510	63.166	179.676
8	51.394	40.562	91.956
16	45.087	33.454	78.541
24	26.967	28.755	55.722
32	31.925	34.283	66.208
48	23.288	35.895	59.183
64	18.792	34.040	52.832

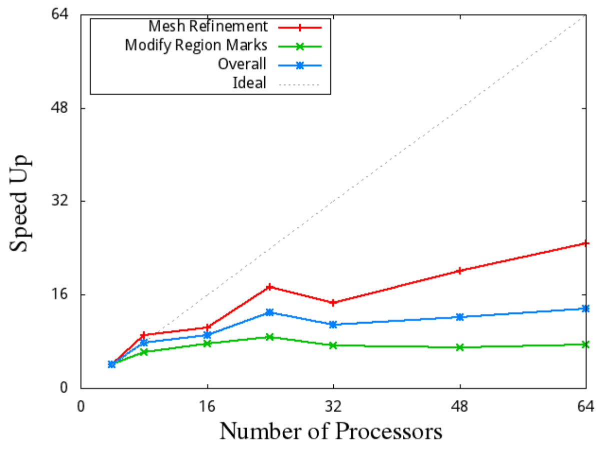


Figure 13: A Dengue virus model: speedups obtained for the whole mesh generation process.

as the number of processors grows. Fortunately, the time is still manageable because the process of modifying region marks is only done to the final mesh. The overall speedup is 13.6 when the number of processes is 64, which is also a good result.

4 Conclusion

This paper provides another approach for tetrahedral mesh generation besides our previously developed standalone tool chain [37] for biomolecules. The mesh generation method takes advantage of the adaptive refinements interfaces of PHG. Moreover, `phgSurfaceCut` in PHG is used to approximate the molecular Gaussian surface to improve the refined meshes and is also used to set the membrane for ion channels. An effective marking strategy is also carried out to ensure the topological correctness. Our mesh generation method is successfully applied to a sphere cavity model, a DNA fragment, a gramicidin A channel and a huge virus molecular system 1K4R. The mesh generation method is validated by the calculated results from FE solutions of the PBE, including electrostatic potential and solvation energy. The results of numerical experiments also show good scalability of our parallel mesh generation algorithm. The most important advantage is that the meshing process and finite element computing can be integrated in the same program for convenient application for biomolecular simulations. In the future, the geometric flow based methods can be taken into consideration for further improvement of the quality of the generated meshes.

Acknowledgments

We would like to thank Rui Diao for his great help. Y. Xie and L.B. Zhang are supported by National 973 Project of China (2011CB309703), National 863 Project of China (2012AA01A309), China NSF (91430215, 91530323 and 11321061), the State Key Laboratory of Scientific/Engineering Computing, and National Center for Mathematics and Interdisciplinary Sciences of Chinese Academy of Sciences. B. Tu, T.T. Liu and B.Z. Lu are supported by the State Key Laboratory of Scientific/Engineering Computing, National Center for Mathematics and Interdisciplinary Sciences, the Chinese Academy of Sciences, the China NSF (91230106, 91530102, 21573274).

A Appendix

Protein Data Bank (PDB) format is a standard for files containing atomic coordinates, which is used for structures in the Protein Data Bank. PQR format is a modification of the PDB format that allows users to add charge and radius parameters to existing PDB data while keeping it in a format amenable to visualization with standard molecular graphics programs. We read the data on a per-line basis from PQR files, the field names and descriptions are given in Table 4.

Table 5 shows partial lines of the DNA fragment PQR file. It has 778 atoms in total and the field of chainID is omitted here.

Table 4: Field names and descriptions of PQR format.

recordName	Type of PQR entry, either be ATOM or HETATM
serial	Atom index
atomName	Atom name
residueName	Residue name
chainID	Chain ID of the atom, optional
residueNumber	Residue index
X Y Z	Three floats which provide the atomic coordinates
charge	Atomic charge (in electrons)
radius	Atomic radius (in A)

Table 5: PQR file of the DNA fragment.

ATOM	1	O5′	G	1	-0.799	8.456	10.751	-0.6223	1.7210
ATOM	2	C5′	G	1	-0.046	9.672	10.891	0.0558	1.9080
ATOM	3	C4′	G	1	1.439	9.368	10.951	0.1065	1.9080
ATOM	4	O4′	G	1	1.747	8.741	12.221	-0.3548	1.6837
ATOM	5	C3′	G	1	1.947	8.370	9.911	0.2022	1.9080
ATOM	6	O3′	G	1	2.202	8.995	8.651	-0.5246	1.6837
ATOM	7	C2′	G	1	3.195	7.823	10.581	0.0670	1.9080
ATOM	8	C1′	G	1	2.760	7.766	12.041	0.0191	1.9080
ATOM	9	N9	G	1	2.216	6.444	12.461	0.0492	1.8240
ATOM	10	C8	G	1	0.910	6.017	12.481	0.1374	1.9080
ATOM	11	N7	G	1	0.758	4.782	12.911	-0.5709	1.8240
ATOM	12	C5	G	1	2.061	4.364	13.181	0.1744	1.9080
ATOM	13	C6	G	1	2.535	3.124	13.671	0.4770	1.9080
ATOM	14	O6	G	1	1.905	2.116	13.971	-0.5597	1.6612
ATOM	15	N1	G	1	3.931	3.131	13.811	-0.4787	1.8240
ATOM	16	C2	G	1	4.760	4.200	13.511	0.7657	1.9080
ATOM	17	N2	G	1	6.065	4.014	13.711	-0.9672	1.8240
ATOM	18	N3	G	1	4.310	5.370	13.051	-0.6323	1.8240
ATOM	19	C4	G	1	2.956	5.375	12.911	0.1222	1.9080
ATOM	20	H5′	G	1	-0.248	10.325	10.029	0.0679	1.3870
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
ATOM	768	H1′	C	12	8.785	-0.021	15.486	0.2029	1.2870
ATOM	769	H2′	C	12	7.636	0.310	17.963	0.0972	1.3870
ATOM	770	H3′	C	12	7.383	-2.032	18.406	0.0615	1.3870
ATOM	771	H6	C	12	6.426	-2.714	16.075	0.1958	1.4090
ATOM	772	H5	C	12	4.059	-2.603	15.413	0.1928	1.4590
ATOM	773	H41	C	12	2.570	0.576	14.223	0.4234	0.6000
ATOM	774	H42	C	12	2.305	-1.155	14.564	0.4234	0.6000
ATOM	775	O2′	C	12	9.754	-0.064	17.086	-0.6139	1.7210
ATOM	776	H5″	C	12	9.009	-2.991	17.238	0.0679	1.3870
ATOM	777	H3T	C	12	8.648	-1.160	20.194	0.4376	1.2000
ATOM	778	H2″	C	12	9.723	0.799	16.668	0.4186	1.2000

References

[1] O.S. Andersen. Gramicidin channels. *Annu. Rev. Physiol.*, 46:531–548, 1984.

[2] D. Bashford. Scientific computing in object-oriented parallel environments. *Lecture notes in computer science*, 1343:233–240, 1997.

[3] P.W. Bates, G.W. Wei, and S. Zhao. Minimal molecular surfaces and their applications r applications. *Journal of Computational Chemistry*, 29(3):380–391, 2008.

[4] A.H. Boschitsch, M.O. Fenley, and H. Zhou. Fast boundary element method for the linear poisson-boltzmann equation. *J. Phys. Chem. B.*, 106(10):2741–2754, 2002.

[5] P.M. Campbell, K.D. Devine, J.E. Flaherty, L.G. Gervasio, and J.D. Teresco. Dynamic octree load balancing using space-filling curves. Technical Report CS-03-01, 2003.

[6] M. Chen and B. Lu. Tmsmesh: a robust method for molecular surface mesh generation using a trace technique. *J. Chem. Theory Comput.*, 7(1):203–212, 2010.

[7] M. Chen, B. Tu, and B. Lu. Surface triangular mesh and volume tetrahedral mesh generations for biomolecular modeling. In *Image-Based Geometric Modeling and Mesh Generation*, pages 85–106. Springer, 2013.

[8] Z. Chen, Y. Xiao, and L. Zhang. The adaptive immersed interface finite element method for elliptic and maxwell interface problems. *J. Comput. Phys.*, 228(14):5000–5019, 2009.

[9] H.L. Cheng and X. Shi. Quality mesh generation for molecular skin surfaces using restricted union of balls. *Computational Geometry*, 42(3):196–206, 2009.

[10] I.L. Chern, J.G. Liu, and W.C. Wang. Accurate evaluation of electrostatics for macromolecules in solution. *Methods Appl. Anal.*, 10(2):309–328, 2003.

[11] L.P. Chew, N. Chrisochoides, and F. Sukup. Parallel constrained delaunay meshing. *ASME APPLIED MECHANICS DIVISION-PUBLICATIONS-AMD*, 220:89–96, 1997.

[12] M.L. Connolly. Molecular surfaces: A review. *Network Science*, 14, 1996.

[13] H.L. De Cougny and M.S. Shephard. Parallel refinement and coarsening of tetrahedral meshes. *International Journal for Numerical Methods in Engineering*, 46(7):1101–1125, 1999.

[14] H.L. De Cougny, M.S. Shephard, and C. Ozturan. Parallel three-dimensional mesh generation. *Computing Systems in Engineering*, 5(4):311–323, 1994.

[15] T.J. Dolinsky, J.E. Nielsen, J.A. McCammon, and N.A. Baker. PDB2PQR: an automated pipeline for the setup, execution, and analysis of poisson-boltzmann electrostatics calculations. *Nucleic. Acids. Res.*, 32:W665–W667, 2004.

[16] H. Edelsbrunner. Deformable smooth surface design. *Discrete & Computational Geometry*, 21(1):87–115, 1999.

[17] Q. Fang. Iso2mesh: a 3D surface and volumetric mesh generator for matlab/octave, 2010.

[18] P.J. Frey. Medit: An interactive mesh visualization software. *Technical Report RT-0253*, 2001.

[19] M. Gerstein, F.M. Richards, M.S. Chapman, and M.L. Connolly. Protein surfaces and volumes: measurement and use. In *International Tables for Crystallography Volume F: Crystallography of biological macromolecules*, pages 531–545. Springer, 2001.

[20] M.J. Holst. The poisson-boltzmann equation: analysis and multilevel numerical solution. 1994.

[21] Y. Ito, A.M. Shih, A.K. Erukala, B.K. Soni, A. Chernikov, N.P. Chrisochoides, and K. Nakahashi. Parallel unstructured mesh generation by an advancing front method. *Mathematics and Computers in Simulation*, 75(5):200–209, 2007.

[22] G. Karypis and V. Kumar. *Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0.* Citeseer, 1995.

[23] R.E. Koeppe and O.S. Anderson. Engineering the gramicidin channel. *Annu. Rev. Cell. Dev. Biol.*, 25(1):231–258, 1996.

[24] N.G.H. Kruithofand and G. Vegter. Meshing skin surfaces with certified topology. In *Computer Aided Design and Computer Graphics, 2005. Ninth International Conference on*, pages 6–pp. IEEE, 2005.

[25] S.S. Kuo, M.D. Altman, J.P. Bardhan, B. Tidor, and J.K. White. Fast methods for simulation of biomolecule electrostatics. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 466–473. ACM, 2002.

[26] R. Löhner. A parallel advancing front grid generation scheme. *International Journal for Numerical Methods in Engineering*, 51(6):663–678, 2001.

[27] B. Lu, D. Zhang, and J.A. McCammon. Computation of electrostatic forces between solvated molecules determined by the poisson–boltzmann equation using a boundary element

method. *J. Chem. Phys.*, 122:214102, 2005.

[28] B. Lu, Y. Zhou, M.J. Holst, and J.A. McCammon. Recent progress in numerical methods for the Poisson-Boltzmann equation in biophysical applications. *Commun. Comput. Phys.*, 3(5):973–1009, 2008.

[29] B. Lu, Y. Zhou, G.A. Huber, S.D. Bond, M.J. Holst, and J.A. McCammon. Electrodiffusion: A continuum modeling framework for biomolecular systems with realistic spatiotemporal resolution. *J. Chem. Phys.*, 127(13):135102, 2007.

[30] B. Lu, M.J. Holst, J.A. McCammon, and Y.C. Zhou. Poisson-Nernst-Planck equations for simulating biomolecular diffusion-reaction processes I: Finite element solutions. *J. Comput. Phys.*, 229(19):6979–6994, 2010.

[31] P. Mehrotra, J. Saltz, and R. Voigt. *Unstructured scientific computation on scalable multiprocessors*. MIT Press, 1992.

[32] S.J. Owen. A survey of unstructured mesh generation technology. In *IMR*, pages 239–267, 1998.

[33] W. Rocchia, E. Alexov, and B. Honig. Extending the applicability of the nonlinear poisson-boltzmann equation: Multiple dielectric constants and multivalent ions. *J. Phys. Chem. B.*, 105(28):6507–6514, 2001.

[34] R Said, N.P. Weatherill, K. Morgan, and N.A. Verhoeven. Distributed parallel delaunay mesh generation. *Computer methods in applied mechanics and engineering*, 177(1):109–125, 1999.

[35] A.I. Shestakov, J.L. Milovich, and A. Noy. Solution of the nonlinear poisson–boltzmann equation using pseudo-transient continuation and the finite element method. *J. Colloid. Interf. Sci.*, 247(1):62–79, 2002.

[36] H. Si. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software*, 41(2):Article 11, 2015.

[37] B. Tu, M. Chen, Y. Xie, L. Zhang, B. Eisenberg, and B. Lu. A parallel finite element simulator for ion transport through three-dimensional ion channel systems. *Journal of Computational Chemistry*, 34(24):2065–2078, 2013.

[38] J. Weiser, P.S. Shenkin, and W.C. Still. Optimization of gaussian surface calculations and extension to solvent-accessible surface areas. *Journal of computational chemistry*, 20(7):688–703, 1999.

[39] D. Xie and S. Zhou. A new minimization protocol for solving nonlinear Poisson-Boltzmann mortar finite element equation. *BIT Numerical Mathematics*, 47(4):853–871, 2007.

[40] Y. Xie, J. Cheng, B. Lu, and L. Zhang. Parallel adaptive finite element algorithms for solving the coupled electro-diffusion equations. *Molecular Based Mathematical Biology*, 1:90–108, 2013.

[41] A. Zaharescu, E. Boyer, and R.P. Horaud. Transformesh: a topology-adaptive mesh-based approach to surface evolution. *In Proceedings of the Eighth Asian Conference on Computer Vision*, II:166–175, November 2007.

[42] B. Zhang, B. Peng, J. Huang, N.P. Pitsianis, X. Sun, and B. Lu. Parallel AFMPB solver with automatic surface meshing for calculation of molecular solvation free energy. *Computer Physics Communications*, 190:173–181, 2015.

[43] L. Zhang. A parallel algorithm for adaptive local refinement of tetrahedral meshes using bisection. *Numer. Math. Theor. Meth. Appl.*, 2(1):65–89, 2009.

[44] Y. Zhang, G. Xu, and C. Bajaj. Quality meshing of implicit solvation models of biomolecular structures. *Computer Aided Geometric Design*, 23(6):510–530, 2006.