Fast algorithms for computing the effective properties of cancellous bone from micro-CT scans

Wai-Yip Chan, the Chinese University of Hong Kong

Advisor: Prof. Miao-Jung Yvonne Ou, University of Delaware

1. Abstract:

Cancellous bone is a two-phase composite with solid trabeculae and bone marrow. Effective properties of cancellous bones are important in assessing bone health. Computation of the effective properties is carried out by solving partial differential equations in the domain constructed from micro-CT scans. Due to the very complex microstructure of the trabeculae, traditional meshing softwares fail to handle the meshing task. A state-of-the-art 3D segmentation algorithm was applied to a stack of micro-CT scans of cancellous bone, followed by the construction of the signed distance function of the 3D structure. We developed a meshing algorithm modified from Matlab-based DISTMESH, which can efficiently generate high-quality boundary mesh for the cancellous bone. Finally, the partial differential equations were solved by the Boundary Element Method accelerated by the Fast Multipole Method, which is an O(N) fast algorithm. Our results show that the mesh is of high quality and that we can apply boundary element method for solving PDEs with our mesh.

2. Introduction

For best performance of the BEM-FMM on the three-dimensional bone Ω, we apply what is called distmesh [1], introduced by P.-O. Persson, to generate a high-quality mesh on $\partial\Omega$. There are several versions of distmesh. In this paper, we simply refer to distmesh as the version, distmeshsurface, which is to generate a mesh on a closed surface.

Distmesh requires us to input a signed distance function $\phi(x)$, a size function $h(x)$, grid size h_0 and a closed and bounded cuboid R, where $\Omega \subset R \subset \mathbb{R}^3$. The sign convention for ϕ used in distmesh is negative inside Ω and positive outside. The size function h is to specify the distribution of the edge length. h_0 and R is used to generate initial mesh. Distmesh contains four main steps: generating initial mesh, retriangulation, movement step and projection step.

Distmesh firstly generates a grid in R with grid size h_0 and then applies the function "isosurface" in Matlab to generate an initial mesh for the surface $\partial\Omega$ according to the zero set of ϕ. Unlike the convention used in "isosurface", distmesh reverses the sign convention of ϕ. As a result, the orientation of the triangles points inwards. Since what we want to mesh is the zero set of ϕ, one can use $-\phi$ as an input for distmesh to obtain almost the same

mesh with orientation of triangles pointing outward.

Let p be the set of all node points, $p = \{p_i | \ i = 1, \dots, N\}$, an N-3 array. Regarding every point p_i as a truss and every edge of triangles as a bar, we now go to the movement step. The algorithm is to find the equilibrium $\mathrm{F}(p) = 0$ by introducing an artificial time-dependence and by the forward Euler method, that is

$$\text{Movement Step: } p^{(\mathrm{n}+1)} = p^{(\mathrm{n})} + \delta \mathrm{t}\, \mathrm{F}(p^{(\mathrm{n})}) \tag{1}$$

where $\mathrm{F}(p^{(\mathrm{n})}) = \{F(p_i^{(\mathrm{n})}) | \ i = 1, \dots, N\}$ is an N-3 array and $F(p_i^{(n)})$ is the sum of all forces acting on the node p_i in the n-th iteration. The magnitude of the force f on each bar is defined by

$$\mathrm{f}(\ell) = \begin{cases} \mathrm{k}(\ell_0 - \ell) \text{ if } \ell < \ell_0 \\ 0 \quad \text{if } \ell \geq \ell_0 \end{cases} \tag{2}$$

where ℓ is the current length of the edge, ℓ_0 is proportional to the size function h(mid-point of ℓ) and k is set to be 1. The force is a repulsive force acting on the nodes.

After the movement step, all the points will no longer stay on $\partial\Omega$. To bring every point back to $\partial\Omega$, it applies the signed distance function ϕ by the following equations:

$$\text{Projection Step: } \begin{cases} p^* = p + \delta p \\ \delta p = -\phi(p) \frac{\nabla\phi(p)}{|\nabla\phi(p)|} \end{cases} \tag{3}$$

where p^* is the points after projection. we evaluate $\nabla\phi$ by

$$\phi_{\mathrm{x}} = \frac{\phi(\mathrm{x}+\delta\mathrm{x}) - \phi(\mathrm{x})}{\delta\mathrm{x}} \tag{4}$$

and similar for ϕ_{y} and ϕ_{z}, where $\delta\mathrm{x} := \mathrm{h}_0\sqrt{\mathrm{eps}} \ll \mathrm{h}_0$ and eps is the double precision in Matlab.

However, distmesh is not generally capable of generating a mesh on any surface. Due to the bad initial mesh by “isosurface” and the retriangulation in distmesh, which only considers the qualities of the triangles, distmesh always fails for the surface of geometrically complicated object. But even for generating a mesh on a unit sphere, there are some cases when distmesh fails. Figure 1 shows the failure of distmesh on a unit sphere after 70 iterations with retriangulation in each iteration. In the figure, one triangle overlaps another triangle and the system (equation (1)) will never be convergent.

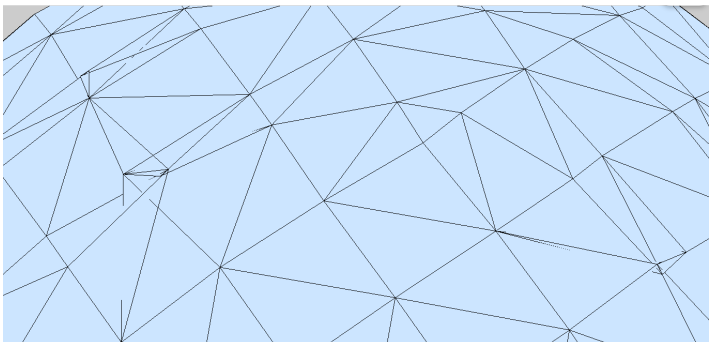

Figure 1 Unit sphere with $R = [-1.2,1.2]^3, \mathrm{h}_0 = 0.2$ and $\delta \mathrm{t} = .001$

In our modified mesh, we solve the problem shown in figure 1 by fixed points. Our efforts mainly focus on overlapping triangles on the curves $\partial(\partial\Omega \cap \partial R)$, where the signed distance function is not smooth. Two figures below indicate how the mesh breaks at the top of a cylinder. In figure 2(a), it is clear that the blue point is going to the right while the red is going to the left. Then, after thirty more iterations, it goes to figure 2(b), where there are overlapping triangles in the mesh.

Since overlapping triangles always occurs on the curves $\partial(\partial\Omega \cap \partial R)$, to avoid this, we divide the mesh into two parts: open mesh on $\partial\Omega \backslash \partial R$ and meshes on $\partial\Omega \cap \partial R$. The open mesh fixes all the points on ∂R generated by the initial mesh while the meshes on $\partial\Omega \cap \partial R$ applies the function "distmesh2d" [1], also developed by P.-O. Persson, with fixed points on ∂R inherited from the open mesh. Distmesh2d is another version of distmesh which is to generate a mesh on a closed and bounded set in a plane. We take advantage of distmesh2d to generate a mesh on each planar surface of ∂R. Another advantage for dividing the mesh into two parts is that we apply BEM-FMM alternatively on two meshes to compute the effective properties of Ω by solving PDE. We will look into the PDE later in this paper.

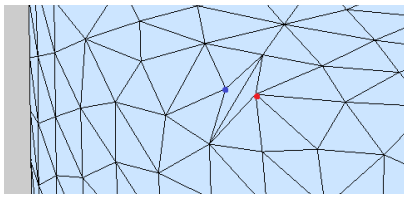

Figure 2(a)

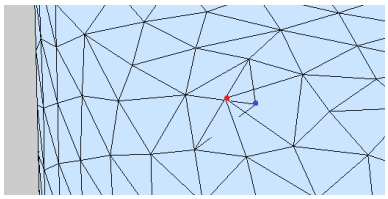

Figure 2(b)

3. Modified distmesh

Let $\Omega_1 \subset R' \subset \mathbb{R}^3$ be a bone as a closed and bounded object. Define the void space $\Omega_2 := \overline{R' \backslash \Omega_1}$ and the interface $I := \Omega_1 \cap \Omega_2$ of Ω_1 and Ω_2. Our first step is to generate a mesh on I. We start with the discretized signed distance function D_I of I given on a 3-dimensional Cartesian grid in R' with grid size h_0'. The sign convention is

$$\mathrm{D}_I(\mathrm{p}) \begin{cases} < 0 & if\ p \in \Omega_1 \backslash I \\ = 0 & \text{if } \mathrm{p} \in I \\ > 0 & \text{if } \mathrm{p} \in \Omega_1 \backslash I \end{cases}.$$

For generating the signed distance function, refer to redistancing in [2]. We take $R := R'$ and $\mathrm{h}_0 := \mathrm{h}_0'$ as inputs of distmesh and take $\phi: R \to \mathbb{R}$ to be the signed distance function obtained by linear interpolation with respect to D_I. Although ϕ should be defined everywhere on $\mathbb{R}^3$, interpolation only gives us the information of ϕ on R. Taking it into account, we have to handle carefully for some points near ∂R. For example, to calculate the value of $\nabla\phi$ at some points x with $\mathrm{x} + \delta\mathrm{x} \notin \mathrm{R}$, we use the formula

$$\phi_{\mathrm{x}} = \frac{\phi(\mathrm{x}) - \phi(\mathrm{x} - \delta\mathrm{x})}{\delta\mathrm{x}} \tag{5}$$

instead of formula (4). As for the size function h, we will adopt it with the curvature of I. The details follow later in this paper.

Firstly, since the mesh does not perform well on small pieces, we first manage to get rid of them from D. We divide the grid points g with $\mathrm{D(g)} \le 0$ into a number of cells, $\mathrm{G}_1, \mathrm{G}_2, \ldots$ with the following properties:

(i) for any $\mathrm{g} \in \mathrm{G}_i, \forall i$, there exists a neighbouring point $\mathrm{g}' \in \mathrm{G}_i$, i. e. $|\mathrm{g}' - \mathrm{g}| = \mathrm{h}_0$

(ii) for any $\mathrm{g} \in \mathrm{G}_i$ and $\mathrm{g}' \in \mathrm{G}_j$, for $i \neq j$, g and g′are not neighbouring points, i. e. $|\mathrm{g}' - \mathrm{g}| > \mathrm{h}_0$

We reject the set $\mathrm{G_i}$ if

$$|\mathrm{G_i}| < m \tag{6}$$

where m is a number defined by the user. Suppose $\mathrm{G_{rej}}$ is the union of all rejected sets. We finally obtain the function $\mathrm{D_{new}}$ defined by

$$\mathrm{D_{new}(g)} = \begin{cases} 1 & \text{if } \mathrm{g} \in \mathrm{G_{rej}} \\ \mathrm{D(g)} & \text{otherwise} \end{cases} \tag{7}$$

Secondly, to deal with the problem shown in figure 1, we look into these triangles. The first idea is by the quality of the triangles, which is defined by

$$\mathrm{q} = 2\frac{\text{radius of inscribed circle}}{\text{radius of circumscribed circle}} = \frac{(\mathrm{b+c-a})(\mathrm{c+a-b})(\mathrm{a+b-c})}{\mathrm{abc}} \tag{8}$$

where a,b,c are the edge length of the triangle. Before a triangle overlaps another, it is sometimes of very bad quality (q<0.1). However, we cannot use it as a criterion because the initial mesh always contains bad triangles of $\mathrm{q} \approx 0$. Instead, we consider the orientation of triangles. For convergent system, distmesh gives a coherent orientation of triangles all pointing outwards (with our sign convention). In figure 3(a), it shows three triangles whose orientation is reversed (compared with figure 1).

With this in mind, we modified distmesh to control these triangles by looking for any triangle with vertices $\{\mathrm{p}_1, \mathrm{p}_2, \mathrm{p}_3\}$ with the following criterion

$$\phi(\mathrm{p}_i) \times (\mathrm{p}_2 - \mathrm{p}_1) \cdot (\mathrm{p}_3 - \mathrm{p}_1) > 0 \text{ for } i = 1,2,3$$

If any triangle satisfies this criterion, the orientation of the triangle is considered to be reversed. If any triangle is reversed, we will run that iteration again with its vertices being fixed. But for some triangles with two points on ∂R, it will never have a reversed orientation unless the only moving point goes to ∂R. In this case, we do not apply the above criterion for that triangle, but fix the moving point until it goes to ∂R. Figure 3(b) shows the resulting mesh with orientation control after 2000 iterations with $\max_i\{\left|p_i^{(\mathrm{n})} - p_i^{(\mathrm{n}-1)}\right|\} \le$5e-5, which shows the convergence of the system. Although this criterion can locate almost all such triangles, some triangles might have already overlapped before we detected it. In such cases, we correct them manually after the mesh.

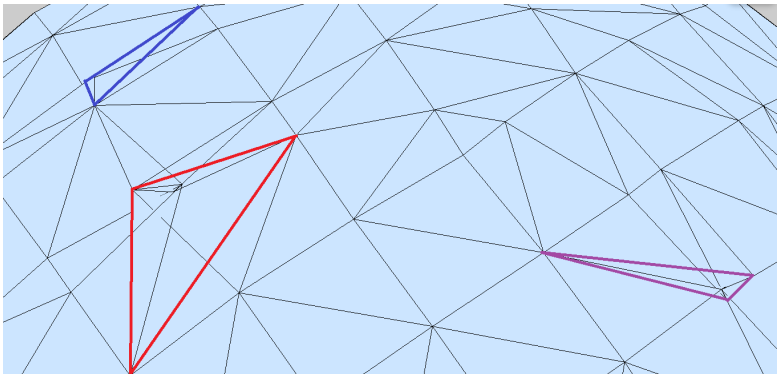

Figure 3(a) triangles of reversed orientation

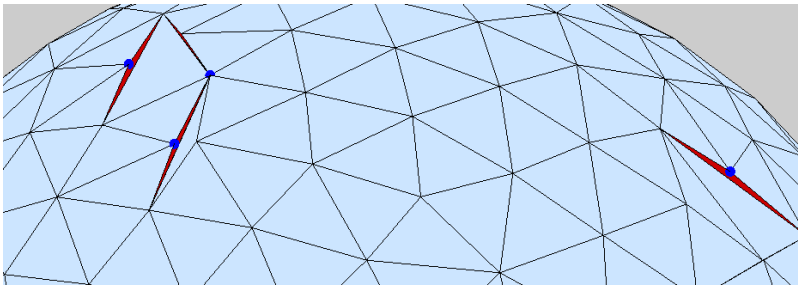

Figure 3(b) convergent modified distmesh by fixed points (marked by Matlab code)

Thirdly, distmesh applies the signed distance function to project any point p_i to the interface I by (3) – (5). However, because of the computing error caused by interpolation in "isosurface" and in calculating $\nabla\phi$, the points after projection are sometimes not very close to I in some iteration. Define the error after n-th projection to be

$$e_n = \frac{\max_i\{|\phi(p_i)|\}}{h_0}$$

In our mesh, after the first movement step, e_1 goes to 0.35 (see figure 4 (a)). The problem is that, in some cases, the algorithm with 1 movement and 1 projection is not convergent, but increasing the projections between movements helps. Since the projection task is very fast in distmesh, more projections will not substantially slow down the speed. Therefore, the modified distmesh runs the projection step several times in each iteration until $e_n < \gamma$ (in our mesh, we take $\gamma := .01$). Another advantage is that, since the discretized distance function ϕ is obtained by redistancing with grid size h_0, when we want to control the number of triangles by changing the value of h_0, it occurs numerical error in interpolation of ϕ. Figure 3(b) shows the error when we choose $h_0 = 1.5h_0'$. In this case, the projection is convergent slower than choosing $h_0 = h_0'$. From the errors, we can also know whether the distance function is suitable for projection in distmesh for any other choice of h_0.

Fourthly, after discovering that the overlapping problem is serious on ∂R (see figure 5(a)), we decide to firstly generate the mesh on I only instead of $\partial\Omega_1$. The idea is also come from the nice results of "distmesh2d". However, the original distmesh does not support for generating a mesh on a surface with boundary. Figure 5(b) shows the resulting mesh when

$e_{intial\ mesh}$	1.5906E-13
$e_{after\ movement}$	0.35500
e_1	0.03410
e_2	0.01210
e_3	0.00074

$e_{intial\ mesh}$	5.4806e-014
$e_{after\ movement}$	0.24760
e_1	0.05420
e_2	0.04440
e_3	0.03480
e_4	0.00500

Figure 4(a) $h_0 = h_0', \max_{\forall i}\{|\delta tF(p_i)|\} = 0.3$ Figure 4(b) $h_0 = 1.5h_0', \max_{\forall i}\{|\delta tF(p_i)|\} = 0.3$

we use $\phi(\mathbf{x}) = \sqrt{x^2 + y^2} - 1$ and $R = [-1,1]^3$ for the original distmesh. The red circles in the figure is on z=1 and z=-1 respectively. Therefore, we have to modify distmesh to be able to obtain a mesh on I. Define $\pi_i, i = 1,..,6,$ be the six planar surface consisted in ∂R and define $q_j = \left\{p_i^{(0)} \in \pi_j \middle| 1 \le i \le N\right\}, j = 1, \dots, 6$. By taking h_0 to be an integer-valued factor of the box lengths, the function "isosurface" will result in a set of edges ℓ_{q_j} on ∂R, for j=1,…,6, with the properties:

$$\forall q_1 \in q_j, \exists q_2 \in q\ s.t.\ |q_1 - q_2| \le \sqrt{2}\, h_0, \text{ and}$$

$$\forall r \in I \cap \partial R, \text{dist}\left(r, \bigcup_{\ell \in \ell_{q_j}} \ell\right) \le \sqrt{2}, j = 1, \dots, 6$$

In the modified distmesh, we fix all the points in q to avoid any distortion similar to what is shown in figure 2. Although some of the points in q_j are very close, e.g. distant from each other about a double precision, we will tackle this problem at the end of the whole mesh by limiting their minimum distance on ∂R. There is another advantage for fixed points on ∂R. If we want to mesh $\partial(\Omega_1 \cap R'')$, for some $R'' \subset R'$, i.e. taking $R := R''$ as input in distmesh, by fixing all the points on $\partial R''$, the moving points always stay away from $\partial R''$ at a small distance (except for several points). Then the union of all small neighborhood (controlled by δt) of moving points are always contained in R''. Therefore, we are able to generate a mesh on $\partial(\Omega_1 \cap \backslash R'')$.

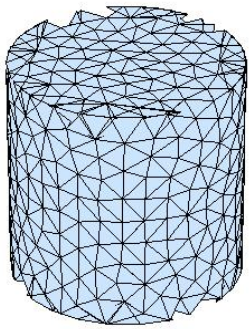

Figure 5 (a) Serious failure on two circumferences

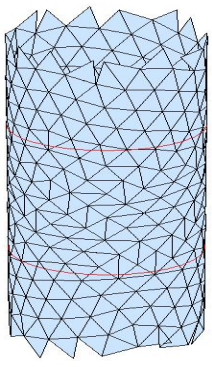

Figure 5(b) Distmesh for $\phi(\mathbf{x}) = \sqrt{x^2 + y^2} - 1$

Fifthly, we generate mesh on $\partial\Omega_1 \backslash I$, or equivalently $\Omega_1 \cap \partial R$, separately on each planar surface π_i of ∂R by the following procedure:

Define P be the set of all node points of the mesh on I.

Define g_i be the set of all grid points on π_i with grid size h_0.

Step1: Save to a set $\mathrm{P}_{\mathrm{fix}}$ all the points $\mathrm{p} \in \mathrm{P}$ satisfying

$$\mathrm{dist}(\mathrm{p}, \pi_i) < \alpha \tag{9}$$

If there are less than 3 points in $\mathrm{P}_{\mathrm{fix}}$, go to Step 9.

Here we take α (we take $\alpha = .01\mathrm{h}_0$ in our mesh) to be the tolerance such that we regard all points p satisfying (9) as points on π_i. This step is essential to help minimizing the number of "holes" in the mesh unless the mesh is actually in equilibrium. We will talk about the "holes" in "Resulting Mesh" in this paper.

Step2: Delete all the triangles with all its vertices $\mathrm{p} \in \mathrm{P}_{\mathrm{fix}}$.

Step3: Save to $\mathrm{P}_{\mathrm{fix}}$ all the points $\mathrm{p} \in \mathrm{g}_i \cap \partial\pi_i$, where $\mathrm{g}_i \cap \partial\pi_i$ is all grid points on the edges of π_i, satisfying $\phi(\mathrm{p}) \leq 0$.

Step4: Numerically compute the signed distance function $\psi_i(\mathrm{x})$ with respect to the region $\Omega_1 \cap \pi_i$ on π_i by interpolation.

Step5: Set the size function to be $\mathrm{h}_i(\mathrm{x}) := 1 + \frac{(\lambda-1)|\psi_i(\mathrm{x})|}{\mathrm{h}_0}$, where λ is the ratio of the expected edge length at distance h_0 from the curve $I \cap \pi_i$ to that on $I \cap \pi_i$

Step6: Generate the mesh on π_i by "distmesh2d" using the distance function $\psi_i(\mathrm{x})$, the size function $\mathrm{h}_i(\mathrm{x})$, grid size h_0, bounding box π_i and fixed points in $\mathrm{P}_{\mathrm{fix}}$. In the mesh, we avoid any points from being too close to the curve $I \cap \pi_i$ by rejecting all the points p with $\psi_i(\mathrm{p}) > -\frac{\mathrm{h}_0}{2}$ in the initial mesh and fixing all the points p with $\psi_i(\mathrm{p}) > -\frac{\mathrm{h}_0}{2}$ during the iterations.

Step 7: Clean up all the triangles with $\mathrm{q} < 0.1$ according to equation (8).

Note: Since "distmesh2d" only rejects triangles with $\phi(\text{centroid of triangles}) \leq 0$, some triangles which is not a part of the mesh of $\partial\Omega_1$ will be formed near the curves $I \cap \pi_i$ on π_i. Luckily, these triangles are always of bad quality, they will be clean up in step 7. After cleaning up bad triangles, there will be many holes in the mesh on $\Omega_1 \cap \pi_i$ because triangles with two close vertices are deleted. This problem will then be tackled by limiting the minimum distance of points on ∂R.

Step 8: Combine the points on $I \cap \pi_i$ with the points in P.

Step 9: Take another planar surface π_j and go to step 1 until meshes on all the six planar surface of ∂R is completed.

Step 10: Take $\overline{\phi} := -\phi$ and follow step 1 to step 9 for the mesh on $\Omega_2 \cap \partial R$.

Lastly, we mentioned that we have to limit the minimum distance of points on ∂R. We do it by collapsing the points on ∂R which are too close. We look for any pair of points $\mathrm{p}_1, \mathrm{p}_2 \in \mathrm{P} \cap \pi_i$ for each i, with $|\mathrm{p}_1 - \mathrm{p}_2| < \frac{\mathrm{h}_0}{3}$, and then convert every triangle with vertices $\{\mathrm{p}_2, \mathrm{p}', \mathrm{p}''\}$ to $\{\mathrm{p}_1, \mathrm{p}', \mathrm{p}''\}$ and delete the triangle $\{\mathrm{p}_1, \mathrm{p}_2, \mathrm{p}'''\}$, where $\mathrm{p}', \mathrm{p}'', \mathrm{p}''' \in \mathrm{P}$. In this step, we do the same operations on the mesh of $\partial\Omega_1$ and $\partial\Omega_2$ simultaneously to assure that the triangles on I are coherent.

4. Curvature Adaptation

To better approximate the surface I with a fixed number of triangles, we use the curvature as a standard for edge length. We hope to have a mesh with more points and smaller edge length in somewhere whose curvature is large, and vice versa. To generate a non-uniform mesh on I, we applied the Gaussian curvature [3] of I to compute the size function h from the signed distance function ϕ. Define:

$$\mathrm{H}(\mathrm{f}) = \nabla(\nabla\phi)$$
$$\mathrm{H}^*(\mathrm{f}) = \left[\mathrm{Cofactor}\left(\mathrm{H}_{\mathrm{ij}}\right)\right]$$
$$\mathrm{K} = \frac{\nabla\phi * \mathrm{H}^*(\phi) * \nabla\phi^{\mathrm{T}}}{|\nabla\phi|^4} \tag{10}$$

where K is the Gaussian curvature. The first order derivatives is approximated numerically as in equation (4) and the second order derivatives are evaluated by

$$\phi_{\mathrm{xx}} = \frac{\phi(\mathrm{x}+\delta\mathrm{x}) - 2\,\phi(\mathrm{x}) + \phi(\mathrm{x}-\delta\mathrm{x})}{(\delta\mathrm{x})^2} \tag{11}$$

$$\phi_{\mathrm{xy}} = \frac{\phi(\mathrm{x}+\delta\mathrm{x},\mathrm{y}+\delta\mathrm{y}) - \phi(\mathrm{x}+\delta\mathrm{x},\mathrm{y}) - \phi(\mathrm{x},\mathrm{y}+\delta\mathrm{y}) + \phi(\mathrm{x},\mathrm{y})}{\delta\mathrm{x}\,\delta\mathrm{y}} \tag{12}$$

and similar formulae for other second order derivatives. Unlike the case for first order derivatives, we should take $\delta\mathrm{x} = \delta\mathrm{y} = \mathrm{h}_0\sqrt[4]{\mathrm{eps}}$ here to get rid of any troubles in $(\delta\mathrm{x})^2$ and $\delta\mathrm{x}\,\delta\mathrm{y}$. As mentioned, ϕ is only defined on R, so we cannot use (11) and (12) for some

points near ∂R. Our solution is to calculate the curvature K at the grid points inside R only and then assign $\mathrm{K(p)} = \alpha$, for all grid points $\mathrm{p} \in \partial R$ (we take $\alpha := 10\text{th percentile of } \mathrm{K(p)}$ for all grid points $\mathrm{p} \in R \setminus \partial R$ Another advantage is to make it easy for the mesh on $\Omega_1 \cap \partial R$ and $\Omega_2 \cap \partial R$, where we defined $\mathrm{h}_i(\mathrm{p}) = \mathrm{const}\ \forall \mathrm{p} \in I \cap \partial R$.

For the discretized distance function D, if we apply linear interpolation to obtain ϕ, the numerator in (11) and (12) will be close to zero since ϕ is linear between grid points. Therefore, we apply cubic interpolation instead of linear interpolation to approximate the second order derivatives. The result is very close to the actual curvature. Figure 6(a) shows the result obtained from the signed distance function $\phi(\mathrm{x}, \mathrm{y}, \mathrm{z}) = \sqrt{\mathrm{x}^2 + \mathrm{y}^2 + \mathrm{z}^2} - 1$ of a unit sphere while figure 6(b) shows the result from the cubic interpolation of the discretized distance function on $[-1,1]^3$ with grid size 0.2. All the samples are from the grid points of the grid on $[-1,1]^3$ with grid size 0.2.

	Curvature
Min. value	1.00000
Max. value	1.00712
Mean	1.00255
Standard deviation	0.00223

Figure 6(a) exact function for unit sphere

	Curvature
Min. value	0.93648
Max. value	1.09932
Mean	1.02019
Standard deviation	0.05049

Figure 6(b) cubic interpolation

The size function h is then defined by

$$\mathrm{h} = \frac{1}{K} + \beta \tag{13}$$

where β is to control the scale of the ratio of the expected length. Hence, at any point, the larger the curvature is, the smaller the value h assumes and the smaller the edge length is near that point.

However, there are many occasions that the difference of the curvatures between two consecutive grid points is large and the same happens for the size function h. In turn, triangles with bad quality will be formed. To prevent this from happening, we applied a grading limiting process on h described in [1]. The process generate a function $\bar{\mathrm{h}}$ satisfying, for any two neighboring points $\mathrm{p_i}$ and $\mathrm{p_j}$, $\bar{\mathrm{h}}(\mathrm{p_i})$ and $\bar{\mathrm{h}}(\mathrm{p_j})$ do not differ by a factor $\mathrm{g} \geq 1$, i.e. $\bar{\mathrm{h}}(\mathrm{p_i}) \leq \mathrm{g}\bar{\mathrm{h}}(\mathrm{p_j})$ provided that $\bar{\mathrm{h}}(\mathrm{p_i}) \leq \bar{\mathrm{h}}(\mathrm{p_j})$. We remark here that the size function h should only be evaluated at the grid points near I and set that at the remaining grid points to be infinity before grading limiting. There is an example of failure when you evaluate h on the whole region R. Let Ω_0 be a unit sphere in $\mathbb{R}^3$ and ϕ_0 be the corresponding signed distance function. Suppose p_ε be the nearest grid point to the origin. WLOG, we assume $\mathrm{p}_\varepsilon := (\varepsilon, 0,0)$, where $0 < \varepsilon \ll \mathrm{h}_0$. Then we have, for the point $\mathrm{p}_0 := (1,0,0)$, $\bar{\mathrm{h}}(\mathrm{p}_0) \approx$

$h(p_\varepsilon)G^{\frac{1}{h_0}} = \frac{1}{\varepsilon^2} g^{\frac{1}{h_0}}$ provided that $h_0 < 1$. Therefore $\bar{h}(p_0)$ will be affected substantially by the location of p_ε, which is not easy to control. Figure 7 below shows the different before and after curvature adaptation.

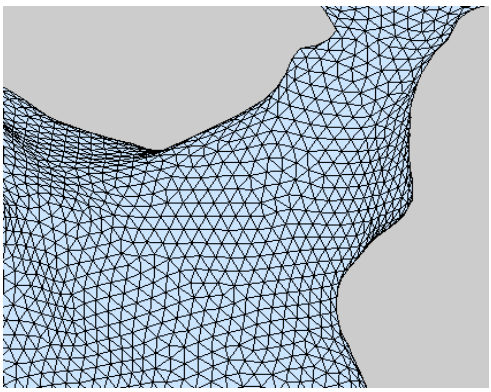

Figure 7(a) Before adaptation

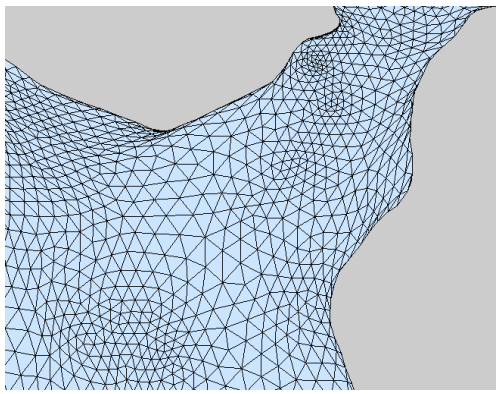

Figure 7(b) After adaptation

5. Resulting Mesh

Since distmesh is well-known for its speed to generate a high-quality triangulation mesh, the most important thing when modifying distmesh is to assure its speed. We compare the time for the original distmesh and the modified one by generating a uniform mesh on a unit sphere with $R = [-1,1]^3$ and $h_0 = 0.01$ for 200 iterations. The number of points is 188142 and the number of triangles is 376280. The original distmesh takes 420s while the modified takes 540s. The result shows that the modified distmesh still has a good performance based on time.

Our experience shows that we should be able to generate the mesh for any choice of mesh size h_0 and any choice of R, provided that $R \subset R'$ and h_0 is an integral factor of the lengths of R. But, for any other choices of h_0 and R, there are more problems in the mesh than that obtained by choosing $h_0 = h_0'$ and $R = R'$.

For our mesh, the discretized signed distance function D is represented by an 145x145x37 matrix with $h_0 = 1.5h_0'$. Since we are generating a non-uniform mesh, we use $\delta t = 0.02$ to decrease overlapping triangles and 2000 iterations to assure a high quality of mesh. It takes 40s to compute the size function h, half an hour to generate the mesh on I, 99s to generate the mesh on $\Omega_1 \cap \partial R$ and $\Omega_2 \cap \partial R$, and 4.6s to control the points on ∂R. The resulting mesh on I contains 26662 points and 51471 triangles.

Although the modified distmesh is well-preformed, there are still holes, due to deleting the triangles with bad quality, and some overlapping triangles in the mesh of I. Luckily, there are totally only ten holes in the mesh $\partial\Omega_1$ and $\partial\Omega_2$, which can be detected by the edge which is shared with only one triangle. As for the overlapping triangles, which have been

detected by orientation, we correct them manually. In our mesh, we totally have 5 such problems

After correcting all these problems in the mesh, we obtain the resulting meshes shown in figure 7 and the triangle qualities (equation (8)) of both meshes are shown in figure 8.

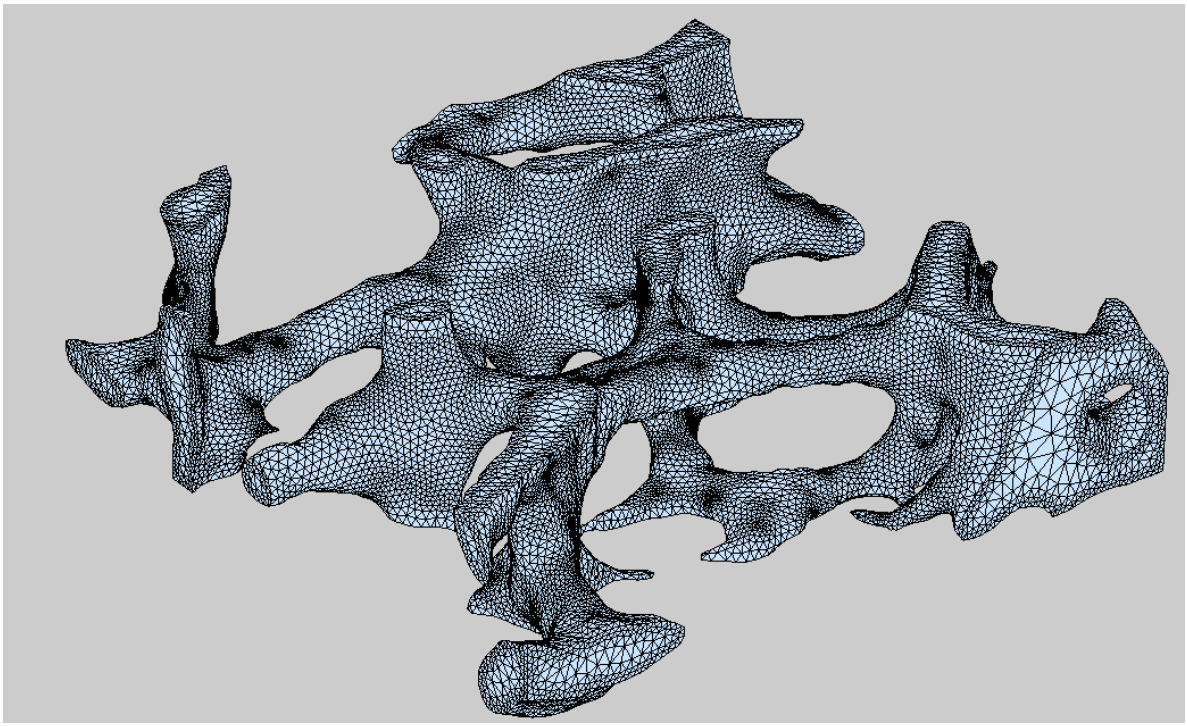

Figure 7(a) Mesh on $\partial\Omega_1$

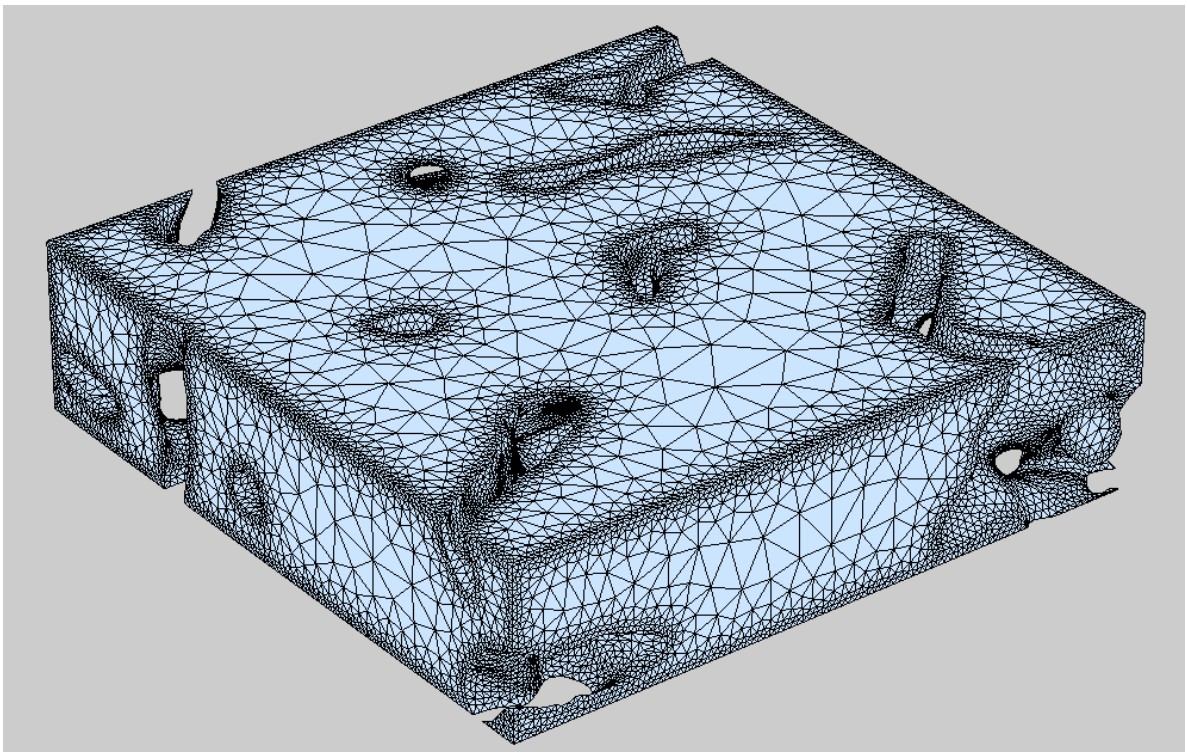

Figure 7(b) Mesh on $\partial\Omega_2$

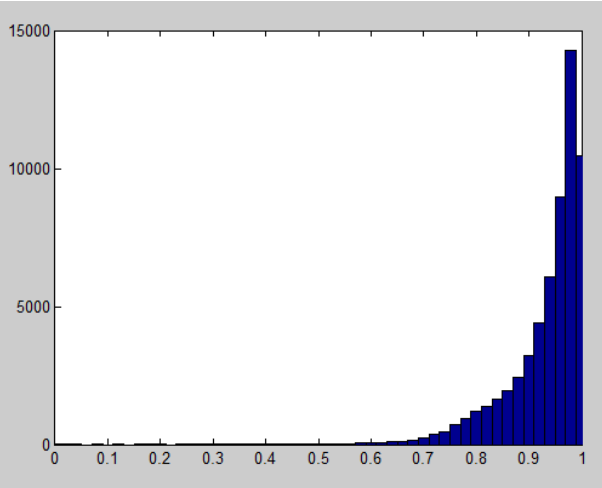

Fig 8(a) Triangle quality of mesh on $\partial\Omega_1$

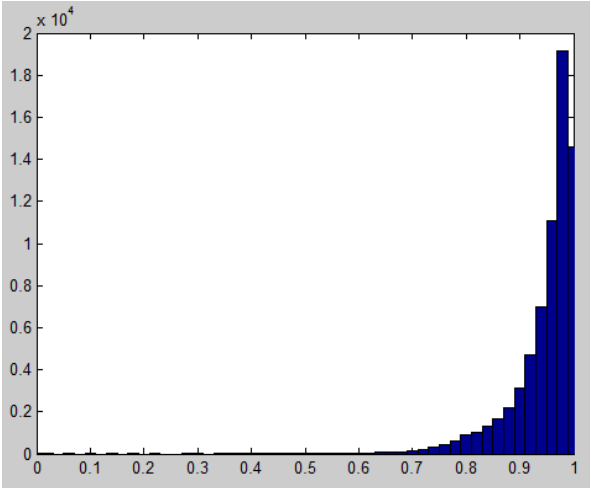

Fig 8(b) Triangle quality of mesh on $\partial\Omega_2$

We apply FMM-BEM, developed by Dr. Yijun Liu, to solve the Laplacian equation $\Delta u = 0$ with Ω_1 as the domain. FMM-BEM is an algorithm to solve the Laplacian equation by Boundary Element Method accelerated by Fast Multipole Method.

Figure 9 below shows the solution of the Laplacian equation with boundary condition

$$\begin{cases} \Delta u = 0 \\ u|_{\partial\Omega_1 \setminus I} = z - \text{coordinate of the centroid of triangles} \in [0,1] \\ \dfrac{\partial u}{\partial \mathbf{n}}|_I = 0 \end{cases}$$

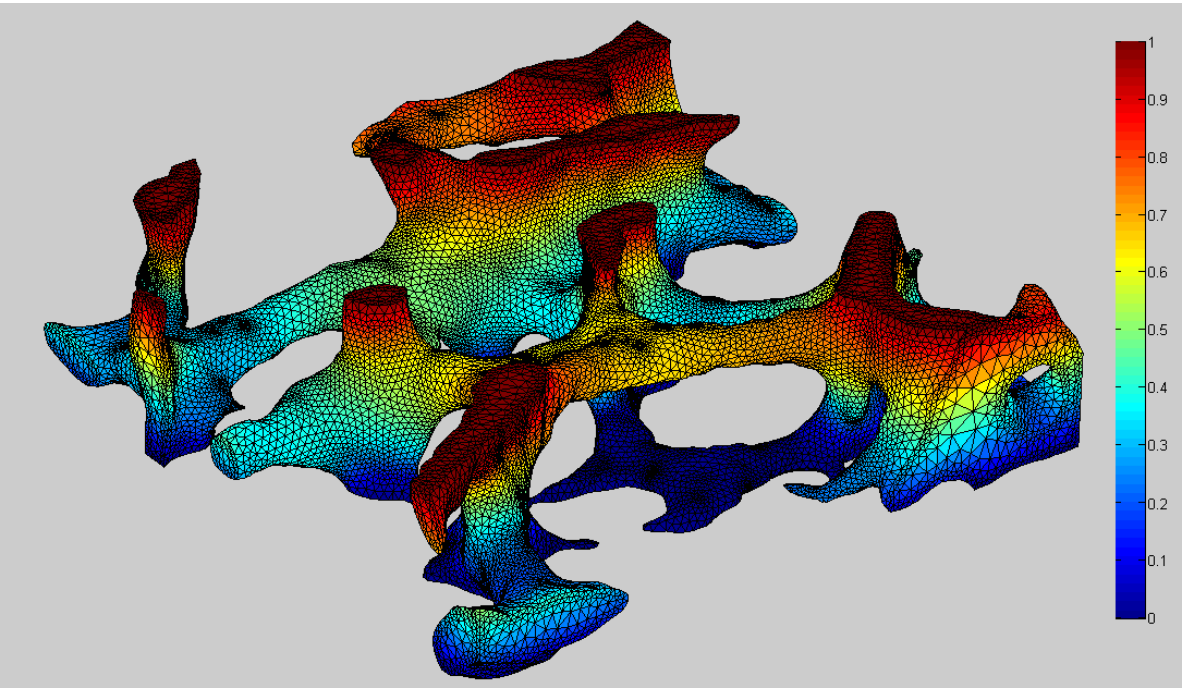

Figure 9

Figure 10 below shows the solution of Laplacian equation with boundary condition

$$\begin{cases} \Delta u = 0 \\ u|_{\pi_1} = 1 \\ \dfrac{\partial u}{\partial \mathbf{n}}|_{\partial\Omega\setminus\pi_1} = 0 \end{cases}$$

where $\pi_1 \in \partial R$ is the plane with the minimum value in x-coordinate of ∂R.

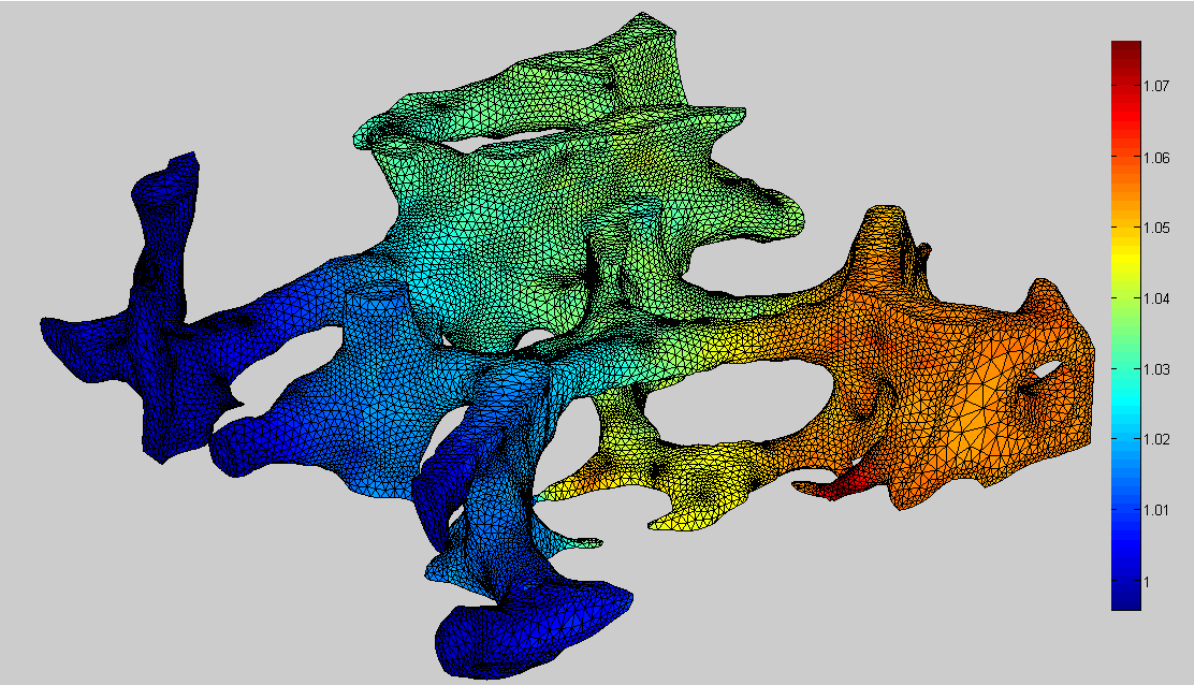

Figure 10

6. Computation of effective properties

Define $\Omega := \Omega_1 \cup \Omega_2$. We use the resulting mesh as an input for the following boundary value problem

$$\begin{cases} \nabla \cdot \left[\left(\epsilon_1 \chi_1(\mathbf{x}) + \epsilon_2 \chi_2(\mathbf{x}) \right) \nabla u \right] = 0 \\ u|_{\partial\Omega} = \sum_{j=1}^{d} E_j^0 x_j \end{cases} \tag{14}$$

where ϵ_1 and ϵ_2 are the dielectric tensors of constituent materials occupying Ω_1 and Ω_2 respectively, and χ_i is the characteristic function of $\Omega_i, i = 1,2$.

The transmission conditions

$$u_1|_I = u_2|_I \tag{15}$$

$$\epsilon_1 \frac{\partial u_1}{\partial \mathbf{n}_1}\Big|_I = -\epsilon_2 \frac{\partial u_2}{\partial \mathbf{n}_2}\Big|_I \tag{16}$$

We solve equations (14)-(16) using FMM-BEM. The iteration details are as follows:

Step 1 Make an initial guess of $\left.\frac{\partial \mathrm{u}_2^{(0)}}{\partial \mathbf{n}_2}\right|_I$. We let $\left.\frac{\partial \mathrm{u}_2^{(0)}}{\partial \mathbf{n}_2}\right|_I \equiv 0$ in this paper. Then solve the Laplacian equation

$$\begin{cases} \Delta \mathrm{u}_2^{(0)} = 0 \\ \left.\dfrac{\partial \mathrm{u}_2^{(0)}}{\partial \mathbf{n}_2}\right|_I := 0 \\ \mathrm{u}_2^{(0)}|_{\partial\Omega_2 \setminus I} = \boldsymbol{E}^0 \cdot \mathbf{x} \end{cases}$$

The solution contains $\mathrm{u}_2^{(0)}|_I$.
Let $n = 1$.

Step 2 Use the condition (15) to solve the Laplacian equation

$$\begin{cases} \Delta \mathrm{u}_1^{(n)} = 0 \\ \mathrm{u}_1^{(n)}|_I = \mathrm{u}_2^{(n-1)}|_I \\ \mathrm{u}_1^{(n)}|_{\partial\Omega_1 \setminus I} = \boldsymbol{E}^0 \cdot \mathbf{x} \end{cases}$$

The solution contains $\left.\frac{\partial \mathrm{u}_1^{(n)}}{\partial \mathrm{n}_1}\right|_I$.

Step 3 Use the condition (16) to solve the Laplacian equation

$$\begin{cases} \Delta \mathrm{u}_2^{(n)} = 0 \\ \left.\dfrac{\partial \mathrm{u}_2^{(n)}}{\partial \mathbf{n}_2}\right|_I = -\dfrac{\epsilon_1}{\epsilon_2}\left.\dfrac{\partial \mathrm{u}_1^{(n)}}{\partial \mathbf{n}_1}\right|_I \\ \mathrm{u}_2^{(n)}|_{\partial\Omega_2 \setminus I} = \boldsymbol{E}^0 \cdot \mathbf{x} \end{cases}$$

The solution contains $\left.\mathrm{u}_2^{(n)}\right|_I$.

Step 4 Let $n = n + 1$ and then go to step 2 until $\left|\mathrm{E}^{(n+1)} - \mathrm{E}^{(n)}\right|$ is less than a tolerance, where $\mathrm{E}^{(n)}$ is the energy norm defined by

$$\mathrm{E}^{(n)} \stackrel{\text{def}}{=} \frac{1}{|\Omega|}\int_\Omega \mathrm{u}^{(\mathrm{n})}\nabla\cdot\phi^{(\mathrm{n})}(\epsilon_1\chi_1 + \epsilon_2\chi_2)\nabla \mathrm{u}^{(\mathrm{n})}]\mathrm{d}\mathbf{x}$$

$$= \frac{1}{|\Omega|}\int_{\partial\Omega_1} \epsilon_1 \mathrm{u}_1^{(n)}\frac{\partial \mathrm{u}_1^{(n)}}{\partial \mathbf{n}_1}\,\mathrm{dS} + \int_{\partial\Omega_2} \epsilon_2 \phi_2^{(n)}\frac{\partial \mathrm{u}_2^{(n)}}{\partial \mathbf{n}_2}\,\mathrm{dS}$$

Also define, for $\mathbf{E}^0 = \mathbf{e}_i, i = 1,2,3,$

$$\left(\mathrm{e}_\mathrm{x}^{(n)}, \mathrm{e}_\mathrm{y}^{(n)}, \mathrm{e}_\mathrm{z}^{(n)}\right)^{\mathrm{T}} \equiv \boldsymbol{\epsilon}_i^* \stackrel{\text{def}}{=} \frac{1}{|\Omega|}\int_\Omega (\epsilon_1\chi_1 + \epsilon_2\chi_2)\nabla \mathrm{u}^{(\mathrm{n})}]\mathrm{d}\mathbf{x}$$

$$= \frac{1}{|\Omega|}\int_{\partial\Omega_1} \epsilon_1 \mathrm{u}_1^{(n)}\mathbf{n}_1\,\mathrm{dS} + \int_{\partial\Omega_2} \epsilon_2 \mathrm{u}_2^{(n)}\mathbf{n}_2\,\mathrm{dS}$$

By the input of our mesh and by taking $\epsilon_1 = 0.3, \epsilon_2 = 1.96$, we obtain the following result for $\mathbf{E}^0 := \mathbf{e_3}$ with tolerance 1e-4. The result shows the convergence of $\mathbf{E}^{(n)}$.

n	e_x	e_y	e_z	E^(n)
1	-0.00098508	0.00683632	1.58587510	1.63682052
2	-0.00644607	0.02487523	1.53420103	1.52199952
3	-0.00278665	0.01540904	1.55885546	1.56840474
4	-0.00524254	0.02037301	1.54528326	1.54004296
5	-0.00360205	0.01779768	1.55334997	1.55577799
6	-0.00469491	0.01910164	1.54832537	1.54547720
7	-0.00396754	0.01847056	1.55155830	1.55187342
8	-0.00445166	0.01875086	1.54942711	1.54754203
9	-0.00412930	0.01864841	1.55085888	1.55039522
10	-0.00434408	0.01866499	1.54988240	1.54841971
11	-0.00420088	0.01868621	1.55055647	1.54976843
12	-0.00429643	0.01865156	1.55008660	1.54882027
13	-0.00423261	0.01868791	1.55041672	1.54948232
14	-0.00427530	0.01865499	1.55018330	1.54901195
15	-0.00424669	0.01868271	1.55034921	1.54934513
16	-0.00424669	0.01868271	1.55034921	1.54934513
17	-0.00426592	0.01866035	1.55023077	1.54910664
18	-0.00425295	0.01867789	1.55031563	1.54927718
19	-0.00426174	0.01866440	1.55025464	1.54915441
20	-0.00425574	0.01867462	1.55029860	1.54924279

Define $\boldsymbol{\epsilon}^* \equiv (\boldsymbol{\epsilon}_1^*, \boldsymbol{\epsilon}_2^*, \boldsymbol{\epsilon}_3^*)$. We finally obtain

$$\boldsymbol{\epsilon}^* = \begin{pmatrix} 1.5711 & 0.0179 & -0.0043 \\ 0.0175 & 1.5193 & 0.0187 \\ -0.0042 & 0.0187 & 1.5503 \end{pmatrix}$$

7. Implementation

7.1. Signed Distance Function

Suppose we have obtained a discretized signed distance function D and the mesh grid $\mathrm{x}, \mathrm{y}, \mathrm{z}$ in R' together with grid size h_0'. We initialize the distance function by the following Matlab code:

```
% 1. Initialize the signed distance function
D=connectivity(D,500);
fd=@(p) dmatrix3d(p,x,y,z,D);
```

```
h0=ep*2;
[x y z]=ndgrid(bbox(1,1):h0:bbox(2,1),bbox(1,2):h0:bbox(2,2),bbox(1,3):h0:bbox(2,3));
D2=reshape(feval(fd,[x(:) y(:) z(:)]),size(x));
fd=@(p) dmatrix3d(p,x,y,z,D2);
```

We wrote a program `Dnew=connectivity(D,m)` to get rid of any small pieces from the discretized distance function. The input `D` is the discretized signed distance function and the input `m` is a number as defined in equation (6). The output is the `Dnew` is a new discretized signed distance function as defined in equation (7). The implementation of this program is fast. It takes 8s to check a matrix with size 145x145x37 with $|\{p: D(p) \leq 0\}|$=142586.

In the code above, we define a new function with respect to another choice of h_0 to get rid of any incoherent in extracting the value from D as a signed distance function by interpolation.

7.2. Size Function

This section is to initialize the size function h from curvature K of the zero set of ϕ. We initialize the size function by the following code.

```
% 2. Initialize the curvature function of the bone
fd2=@(p) dmatrix3dcubic(p,x,y,z,D2);
fhbbox=[bbox(1,:)+h0;bbox(2,:)-h0];   % Only for grid pts on R\(\partial)R
[xx yy
zz]=ndgrid(fhbbox(1,1):h0:fhbbox(2,1),fhbbox(1,2):h0:fhbbox(2,2),fhbbox(1,3):h0:fhbbox(2,3));
nearI=find(abs(feval(fd,[xx(:) yy(:) zz(:)]))<=3*h0);       % Find all pts near I
K=zeros(size(xx));K(nearI)=curvature(fd2,[xx(nearI) yy(nearI) zz(nearI)],h0);  % Calculate the
curvature of those pts near boundary
K=reshape(.5+1./K,size(xx));                          % Equation (10)
K(setdiff(1:numel(K),nearI))=inf;                     % Initialize remaining pts for Grading limiting
alpha=sort(k(nearI));alpha=alpha(floor(numel(alpha)/40));
Ctemp=alpha*ones(size(K)+2);Ctemp(2:end-1,2:end-1,2:end-1)=K;K=Ctemp;       % assign alpha for all
point on \partial R
h=gradlim3d(K,1.2);                                   % Grading limiting
fh=@(p) dmatrix3d(p,x,y,z,h);
```

As mentioned, we use the grid points $p \in \partial R \backslash R$ as input for calculating curvature. The function `K=curvature(fd,p,h0)` simply written according to equations (4),(10)-(14). It requires distance function `fd` (handle), an N-3 matrix of points `p` and grid size `h0` (for computing δx only) as inputs. The output `K` is an N-1 matrix of curvature according to the input `p`. In the

code, we apply cubic interpolation as mentioned for ϕ for the function `curvature` to compute the second order partial derivatives. As for the input `p` of `curvature`, we only choose the grid points near I. Also, we look for a constant α to assign on all points $p \in I \cap \partial R$ before grading limiting.

The function `B = gradlim3d(A,g)` is for grading limiting. It requires a positive 3-dimensional matrix `A` and a factor `g>1` in grading limiting as inputs and the output is a 3-dimensional matrix `B`. The function `gradlim3d` is carried out by:

```
Let heap = all (I, J, K)
For I
    For J
        For K
            Let (i, j, k) satisfy A_ijk = min_{i,j,k}{A_ijk}
            Remove (i, j, k) from heap
            Let (i±, j±, k±) be neighborhood of (i, j, k) ∉ heap
            Set A_ijk = min {A_ijk, gA_{i±j±k±}}
        end
    end
end
```

7.3. Mesh on I

Here we come to the mesh on I. There are several parts modified from `distmeshsurfacenew.m`:

(i) Bring every point going outside R to ∂R and fix all points on ∂R

(ii) Use equation (5) to calculate ϕ_x, ϕ_y, ϕ_z for some points near ∂R instead of equation (4)

(iii) More projections until $e_n < .01$

(iv) Detect if any triangle with reversed orientation, not applied if triangles with 2 vertices on ∂R. Also, mark all triangles fixed by orientation and mark one of its vertex

(v) Save the points and triangles in `autosave.mat` (we can stop the program anytime and load the file to obtain the points and triangles)

The program requires inputs: a signed distance function fd (handle), a size function fh (handle), grid size h0, a 2-by-3 matrix bbox representing the box. The output contains points p, triangles t and a logical matrix, which has the same rows as p, representing the points fixed by orientation. The program will also mark those fixed points in a figure at the end.

7.4. Mesh on ∂R

The mesh on $\Omega_1 \cap \partial R$ and $\Omega_2 \cap \partial R$ are carried out by the the function `[pi,ti]=meshonpartialR(fd,p,t,h0,bbox,sgn)`. The code requires inputs: a signed distance

function `fd`, an N-3 matrix of points `p`, an M-3 matrix of triangles `t`, grid size `h0`, a 2-by-3 matrix `bbox` representing the box and the sign `sgn`, where `sgn=-1` for mesh on $\Omega_1 \cap \partial R$ and `sgn=1` for mesh on $\Omega_2 \cap \partial R$. The output contains an N_i-3 matrix of points `pi`, an M_i-3 matrix of triangles `ti`. The program includes correcting the orientation of the mesh according to input `sgn`.

When we want to obtain both $\Omega_1 \cap \partial R$ and $\Omega_2 \cap \partial R$ simultaneously, we use the following Matlab code.

```
[p1 t1]=meshonpartialR(fd,p,t,h0/1.5,bbox,-1);
[p2 t2]=meshonpartialR(fd,p,t,h0/1.5,bbox,1);
```

7.5. Control on ∂R

Lastly, we control the minimum distance on ∂R. We wrote the code `[p1,t1,p2,t2]=control(h0,bbox,p,p1,t1,p2,t2)` for this purpose. The program requires the inputs: grid size `h0`, a 2-by-3 matrix `bbox` representing the box, an N-3 matrix of points `p` in I, which is used as a standard for collapsing the points, and two matrices of points `p1,p2` and two matrices of triangles `t1,t2`. If one wants to generate one mesh on $\partial\Omega_1$ only, he can take `p2=[]` and `t2=[]`.

8. Future Work

As mentioned in “Resulting Mesh”, the modified mesh still has some holes and some overlapping triangles which are to be corrected manually. Since we desire to assure the speed of the program, we do not have many controls or have a time-consuming retriangulation during the iterations. We hope for finding a fast retriangulation method to replace the original one to decrease the number of overlapping triangles. We also hope for finding a retriangulation method, not required to be fast, that can correct the overlapping triangles locally after the mesh. As for the holes in the mesh, we hope for developing a better algorithm to generate the mesh on ∂R.

9. Bibliography

[1] P.-O. Persson,Mesh Generation for Implicit Geometries.Ph.D. thesis, Department of Mathematics, MIT, Dec 2004

[2] L.-T. Cheng and Y.-H. Tsai. Redistancing by flow of time dependent eikonal equation. J. Comput. Phys., 227(9):4002–4017, 2008.

[3] Ron Goldman. Curvature formulas for implicit curves and surface. Computer Aided Geometric Design 2005;632-658