UNIVERSITY OF CALIFORNIA,
IRVINE

An Interface-Fitted Mesh Generator and Numerical Methods for Elliptic Interface Problems

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Mathematics

by

Min Wen

2018

DEDICATION

To

my parents, my husband and my son

for your love, support and understanding

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I would like to express my deepest appreciation and thanks to my advisor Professor Long Chen for all insightful guidance and continuous supports during my research. I am so lucky to be one of his graduate students since he could always offer countless advice every time whenever I got lost in the research. His scientific attitude, patience and enthusiasm as a advisor have made a profound impact on me. My dissertation would not have been possible without his constant help and guidance in me.

I am grateful to my collaborator Prof. Huayi Wei and he provided me so much help on understanding Virtual Element Methods. Thanks for all enlightening discussions and the efforts made to finish our projects together.

I am also grateful to my dissertation committee members Professor Qing Nie and Professor Hongkai Zhao. I appreciate all the help they have provided for me during the past years. I have learned a lot from them after taking their classes. What's more, my deepest appreciation goes to all the current and past members in the team for all their help.

I am so appreciated to Department of Mathematics at University of California, Irvine for its generous offer of scholarship in the past five years. My dissertation would not exist without its full support.

Last but not least, I am greatly indebted to my family for their selfless support and encouragement, which make my graduate life colorful.

Finally, I would like to thank everyone else who supports me spiritually throughout my graduate life, as well as express my apology that I could not mention personally one by one.

CURRICULUM VITAE

Min Wen

EDUCATION

Doctor of Philosophy in Mathematics **2018**
University of California, Irvine *Irvine, California*

Master of Science in Mathematics **2011**
Wuhan University *Wuhan, China*

Bachelor of Science in Mathematics **2008**
Yangtze University *Jingzhou, China*

RESEARCH EXPERIENCE

Graduate Research Assistant **2013–2018**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Teaching Assistant **2016–2017**
University California, Irvine *Irvine, California*

REFEREED JOURNAL PUBLICATIONS

An Interface-Fitted Mesh Generator and Virtual Element Methods for Elliptic Interface Problems **2017**
Journal of Computational Physics

ABSTRACT OF THE DISSERTATION

An Interface-Fitted Mesh Generator and Numerical Methods for Elliptic Interface Problems

By

Min Wen

Doctor of Philosophy in Mathematics

University of California, Irvine, 2018

Professor Long Chen, Chair

In this thesis, we propose different numerical methods for solving elliptic interface problems in three dimensions and moving boundary problems in two dimensions. In the first part, a simple and efficient interface-fitted mesh algorithm which can produce a semi-unstructured interface-fitted mesh in two and three dimensions quickly is developed in this thesis. Elements in such interface-fitted meshes are not restricted to simplices but can be polygons or polyhedra. Especially in 3D, the polyhedra instead of tetrahedra can avoid slivers, which are the major difficulty in finite element methods. Virtual element methods are applied to solve elliptic interface problems with solutions and flux jump conditions. Algebraic multigrid solvers are used to solve the resulting linear algebraic system. In the second part, we impose the interface-fitted meshes to moving boundary problems and present the applications in biology simulation. Static interface problems with periodic conditions are considered at first, then the higher order accuracy algorithm is also developed. Finally moving interface problems are discussed. We couple the interface Poisson equation and the transport equation for the level set function together to simulate the tissue and tumor growth phenomenon. Numerical results are presented to illustrate the effectiveness of our methods.

Chapter 1

Introduction

We consider finite element methods for solving elliptic interface problems which have a variety of applications in different research fields, including fluid dynamics, material science and biological systems, etc. [23, 63, 73, 83]. The importance of the coupling of the complex geometry of the interface with the numerical methods has been recognized and received rapidly increasing interest in recent years. In this thesis, we discuss the elliptic interface problems at first, and then consider moving boundary problems arising from applications in biology.

1.1 Elliptic Interface Problems

Let Ω be an open and bounded domain in $\mathbb{R}^d (d = 2, 3)$, and Γ be a continuous interface embedded in Ω. The interface Γ separates the domain Ω into disjoint regions Ω^+ and Ω^-, where Ω^+ denotes the exterior domain and Ω^- is the interior domain enclosed by Γ. We consider numerical methods for solving the following elliptic interface problems:

$$-\nabla \cdot (\beta(x)\nabla u(x)) = f(x), x \in \Omega \backslash \Gamma \tag{1.1}$$

with prescribed jump conditions across the interface Γ:

$$[u]_\Gamma = u^+ - u^- = q_0, \tag{1.2}$$

$$[\beta u_n]_\Gamma = \beta^+ u_n^+ - \beta^- u_n^- = q_1, \tag{1.3}$$

and boundary condition:

$$u = g \quad \text{on } \partial\Omega. \tag{1.4}$$

Here u_n denotes the normal derivative $(\nabla u)\cdot n$ with n being the unit norm direction of the interface Γ pointing outward (from Ω^- to Ω^+). The superscripts $+$ and $-$ stand for the restriction of a function on Ω^+ and Ω^-, respectively. The diffusion coefficient $\beta(x)$ is assumed to be uniformly positive and smooth on each subdomain, but may be discontinuous across the interface. Because of that, the solution u is piecewise smooth but the global regularity is low [33, 65, 66].

Numerical methods for elliptic interface problems can be roughly classified into two categories by using either an interface-fitted (also known as body-fitted or interface conforming) mesh or an unfitted mesh (e.g. a uniform Cartesian mesh) in the discretization of the domain. In the unfitted mesh approach, a popular way to enforce the jump conditions is to modify the finite difference stencils or the finite element basis near the interface. A lot of numerical methods in this direction have been proposed such as the immersed boundary method [108], the immersed interface method [78, 79], immersed finite element methods [49, 57, 72, 84], ghost fluid methods [87], matched interface and boundary (MIB) methods [134, 140, 147], multiscale finite element methods [33], extended finite element methods (XFEM) [44, 93, 95], and many others [51, 62, 61, 67, 86, 133]. The jump condition can be also imposed based on the Nitsche's method [99] by introducing penalty terms across interfaces, see, for example, the earlier work by Babuška [5], Barrett and Elliott [8], unfitted FEM by Hansbo and Hansbo [53], hp-discontinuous Galerkin method [92], CutFEM [19, 55], and many others [9, 19, 20, 21, 53, 54, 55, 71, 127, 128]. The most attractive feature of the unfitted mesh approach is the easiness of the mesh generation. Indeed, if the background mesh is Cartesian, there is no need of meshing which is very convenient, especially when the interface is moving in time.

On the other hand, using unfitted mesh approach, it is difficult to capture the complex geometry of the interface and to enforce jump conditions across the interface accurately, and the resulting linear system may not be always symmetric which could cause problems for fast solvers. Furthermore a rigorous error analysis is difficult. Recent progress on immersed finite element methods can be found in [52, 142].

In this work, we focus on the interface-fitted mesh approach. Provided a mesh fitted to the interface, one can use conforming finite element methods and get a symmetric system which can be solved efficiently by fast solvers such as algebraic multi-grid. Rigorous error analysis is possible. Optimal a priori estimates of linear finite element is given in [137, 16, 29] and in [80] for high order finite elements. Recent work using hybridized discontinuous Galerkin (HDG) [68] and weak Galerkin (WG) [98] method is also based on a shape regular and body-fitted triangulation. The challenge of this approach is quickly generating an interface-fitted mesh, especially in three dimensions (3D), which is the topic of this study.

There is a lot of work on the unstructured interface-fitted mesh generation [88, 105, 143]. The unstructured mesh generator is, however, time consuming as it needs to modify the mesh for the whole domain, not just near the interface. For example, extensive and non-trivial computational effort are needed to generate a high quality 3D finite element mesh from biomedical image data or geological image data etc [4, 35].

We are interested in the semi-structured and body-fitted mesh generation methods [12, 14, 110] and will develop a simple and effective mesh generation algorithm. As an illustrative example, to generate an interface-fitted mesh in two dimensions (2D), we start from a uniform Cartesian mesh with N-grid points, and apply three steps: 1) find all the intersection points, the grid points near the interface, and add few auxiliary points; 2) generate a Delaunay triangulation of these points; 3) remove the unnecessary triangles and merge the regular meshes away from the interface. The resulting triangulations can preserve the interface and the maximal angle is bounded by 135°. Since the Delaunay triangulations are only on a local region near the interface, the dominant cost is reduced to $\mathcal{O}(N^{1/2} \log N)$. Due to the semi-unstructured mesh and localization near the interface, some nice properties of structured grids are still preserved such as superconvergence in the energy norm and fast convergence of algebraic multigrid methods [129].

The main restriction of this approach is the quality of the generated mesh especially in 3D. Most finite element methods require discretizing a domain into a set of shape regular tetrahedra in three dimensions. The accuracy of the simulations and the efficiency of the solvers could deteriorate by the presence of badly-shaped elements. The problematic tetrahedra are so-called slivers, which are a type of flat tetrahedra without small edges, but with nearly zero volume. Namely, four vertices of a sliver are almost coplanar. Due to the presence of slivers, three-dimensional mesh generation is much harder than the two-dimensional case, and removing slivers from a 3D tetrahedral mesh is one of the major tasks in the field of mesh generation [39, 81, 97].

We propose a new way to solve this difficulty. We choose polyhedral meshes rather than tetrahedral meshes. Then silvers will be merged into a polyhedron. The shape of the polyhedron or other tetrahedron could be still degenerate but the maximal angle is bounded uniformly away from π. Notice that finite element approximation retains accurate if the maximal angle condition [6] is satisfied. Namely tetrahedra with small volumes are allowed as long as the four vertices are non-planar [1, 38]. Similar results can be established for polyhedral meshes and theoretical justification will be reported somewhere else.

Another difficulty is encountered in the implementation. Due to the large number of possible intersections between the fixed mesh and the interface, a variety of interface-cells are generated leading to an equally large number of treatments, which could result in complex coding logistics; see [109, 110].

We propose an all-in-one solution. The connectedness of intersection points is obtained by the Delaunay algorithm which is a well developed algorithm in computational geometry and efficient implementation is available in many software packages. Our mesh generation algorithm in 3D is similar to the 2D case only different in the step 3: post-processing. The additional work is to merge tetrahedra into polyhedra. To facilitate the merging, the polyhedra are stored in the form of faces and the index of the elements to which the faces belong. The resulting mesh retains these nice properties: the interface is approximately preserved, the maximal angle condition is satisfied, and cost-efficient. The Delaunay algorithm is only called for points near the interface and thus the dominated cost is reduced to $\mathcal{O}(N^{2/3})$ which is considerably smaller comparing with $\mathcal{O}(N)$ assembling and solving of the linear algebraic system. The quality and efficiency of our mesh generation algorithm are balanced and suitable for the finite element simulation. No additional mesh smoothing process is needed in our algorithm. Of course, adding such a mesh smoothing process will furthermore improve the quality of the mesh and probably improve the accuracy of the finite element approximation. However, it will destroy the structure of the grid. In our mesh generator, the background mesh is fixed. The Delaunay algorithm can be called element by element and thus local modification is possible if only part of the interface is changed. These features are important for moving interface problems.

A similar approach was introduced in [50], where the authors introduced the Voronoi diagrams and Delaunay triangulation of a point set of a surface and more focused on the surface mesh generation. Our algorithm seems simpler and more suitable for finite element simulation as we shall discuss below.

Since elements in such interface-fitted meshes are general polyhedra, we shall apply virtual element methods (VEM) [10, 11], which can be considered as an extension of conforming finite element methods to polyhedral meshes. The resulting linear algebraic system is symmetric and positive definite and thus can be solved efficiently using algebraic multigrid solvers. Furthermore, according to our mesh generation algorithm, we will get the polyhedra with triangular and square faces which will be much easier when assembling the matrices in VEM compared to the original approach in [11]. Optimal second order of convergence in the L^2 and L^∞ norms and a super-convergence of energy norm is observed in several numerical examples. Details are presented in Chapter 2 and 3.

1.2 Moving Boundary Problems

Many applications in fluids, materials and biology involve multi-connected domains. In multi-connected problems the boundary between different subdomains can be dealt with moving interfaces such as hot metal forming, mould filling and open channel flows [124, 56, 60, 90]. In numerical simulations, the major difficulties are the position of the moving interface is not known a priori and strongly coupled to the solutions of the flow equations.

Numerical methods for moving interface problems based on the representation of the interface have been developed and they can be roughly classified into two categories. The first category is interface tracking, in which the interface is explicitly represented. The front tracking methods [132, 48, 124] and the marker and cell (MAC) methods [130, 56] belong to this class. Main advantages of the tracking methods are efficient and accurate representation of the interface and simplicity in tracking the motion of particles. However, it is difficult for interfaces with complicated geometry and topological change and particular in three dimensions.

The second category is interface capturing, in which the interface is implicitly embedded in a scalar field function defined on a fixed mesh, such as a Cartesian grid. Two widely used interface capturing methods are the volume of fluid method (VOF) [60] and the level set method [101]. Main advantages of capturing methods are complex interface structures and topological changes which can be captured quite naturally in two and three dimensions; see, e.g., [22, 101, 120]. However, there are also a few disadvantages for these Eulerian approaches. For examples, due to the implicit representation, capturing methods are usually less accurate and less efficient than tracking method in terms of both interface representation, which lead to loss of high order accuracy at the moving front.

The level set method is a popular method in multi-connected domain problems as it is good for computing curvature driven flow problems [120, 119, 125, 123, 91, 90, 126, 32]. There are different approaches to solve the level set equation numerically. For instance, finite difference methods such as forward difference, essentially nonoscillatory (ENO) and weighted essentially non-oscillatory (WENO) schemes are often used [141, 120, 119, 100, 125]. Also discretization of the level set equation by means of finite volume schemes [74, 45, 46] or finite element methods [123, 111, 116] can be found in the literature.

A lot of work has been done for solving partial differential equations with moving interfaces using finite difference methods. For example, immersed boundary methods(IBM) [94], immersed interface method (IIM) [138, 64, 34, 82], a level set/ghost fluid method [89], an embedded boundary formulation [139], and other methods [144, 117].

In Section 4.1, it introduces the tissue growth problems. We start with static interface problems with periodic boundary conditions in one dimension using finite element methods. We introduce another artificial mesh which satisfies the periodic conditions to assemble the matrix. It could avoid for loop in the code to speed up the performance. Then we use Discontinuous Galerkin Method Finite Element Method (DG-FEM) [59] to solve the moving interface problems. While in [103], the approach is transforming the deformed geometry and governing equations to the unit square and then using finite difference methods. We focuses on the spatial discretization of the level set equation by DG-FEM combined with a 4th order Runge Kutta time stepping scheme. In each time step the whole coupled system is solved iteratively. High order accuracy algorithm is also presented.

In Section 4.2, another model tumor growth simulation is presented. Here we focus on solving the transport equation using 5th order WENO and 3rd order Runge Kutta Methods [115] to simulate tumor growth problems.. We could use a set of fifteen nearby points to fit a circle to compute the curvature. And the sign of the curvature depends on the position of the points in the fitted circle. If one point is still in the tumor interface, the curvature is positive. Otherwise, it is negative. For each time step, we use harmonic extension for each component of the velocity, which is different from the EPC methods in [143]. Reinitialization should be taken into consideration if we use signed distance to compute curvature [143], since the level set function will become very flat or steep near the zero level set after several time steps. Reinitialization [120, 104] could be proposed to reshape it to be an signed distance function, which could result in accurate numerical solutions.

Chapter 2

Mesh Generation

We present the interface-fitted mesh generation algorithm in two and three dimensions. Both of them only consist of three steps. Our method is simple and efficient since we just deal with the points near the interface. We also show some properties of the mesh obtained by our algorithm. The interface will be approximately recovered in the triangulation generated by the algorithm and the maximum angle of the triangulation is bounded.

2.1 Interface-fitted Mesh Generator: Two Dimensions

In this section, we introduce our interface-fitted mesh generator in 2D. We first describe the algorithm and then give two examples to illustrate the algorithm. In addition, we prove the generated mesh will preserve the interface and satisfy the maximal angle condition.

2.1.1 Algorithm

Let Γ be an interface embedded in a rectangular domain Ω. Assume Γ can be represented by the zero-level set of a function $\phi(x)$, i.e., $\Gamma = \{x \in \Omega : \phi(x) = 0\}$. The interface Γ separates Ω into subdomains $\Omega^+ := \{x \in \Omega : \phi(x) > 0\}$ and $\Omega^- := \{x \in \Omega : \phi(x) < 0\}$. Note that Ω^- could have multiple connected components when Γ consists of two or more closed curves.

One can easily generate a uniform Cartesian mesh Ω_h of Ω with a given mesh size h. A vertex p of Ω_h is said to be inside if $\phi(p) < 0$, outside if $\phi(p) > 0$, or on Γ if $\phi(p) = 0$; an edge (p_1, p_2) is called a cut edge if $\phi(p_1)\phi(p_2) < 0$; the point which the cut edge intersects with Γ is called an intersection point; a square element K of Ω_h which intersects with the interface Γ, i.e. $|\bar{K} \cap \Gamma| \neq \varnothing$, is called an interface element. We can find interface elements by using one of the following two rules:

1. There exists at least two vertices p and q with opposite sign, i.e., $\phi(p)\phi(q) < 0$;
2. There exists at least two vertices on the interface, namely the value of ϕ on these vertices is 0.

These two rules could detect all the interface elements in Fig 2.1 except case (3), which could be avoided by choosing the initial mesh size h small enough. For disconnected interfaces (cases $(6) - (9)$), we assume it is described by two level set functions (c.f. Example 2.2), and the intersection points can be found by treating each level set function one by one. See Fig 2.1 for the illustration. We remark that it is much more difficult to modify stencils or the basis for such cases. In general, the modified finite difference stencils or modified finite element basis near the interface is to introduce additional but local degrees of freedom near the interface and then use the jump conditions to eliminate these degree of freedom by solving a small linear system element-wise [135, 2, 13, 131, 43, 70, 96]. In almost all of these work, it is assumed the intersection meets the edges of an interface element at no more than two intersections and intersects at different edges for one element, c.f. [142, 57, 86, 85]. If this condition is violated, such as those cases $(6) - (9)$ in Fig 2.1, the local system will be much more involved since it depends how the interface cuts the elements.

We define the *interface points* as the collection of intersection points, vertices of interface elements, and some auxiliary points explained below. When the intersection points are diagonal, we need to add the midpoints of corresponding elements, which are called auxiliary points.

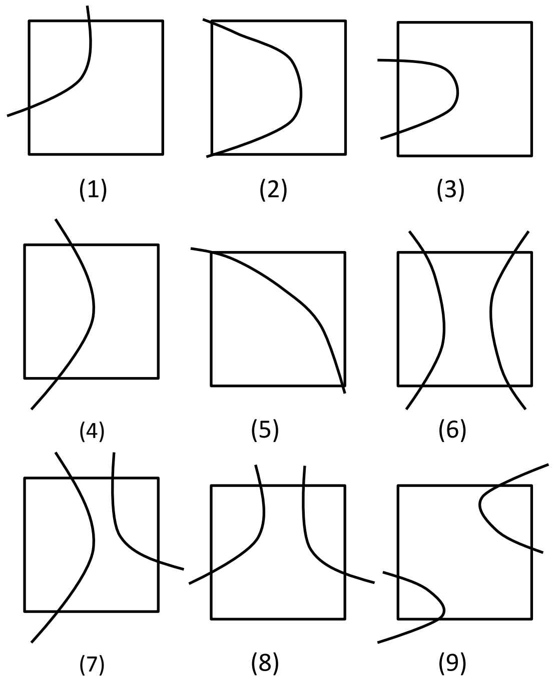


Figure 2.1: Example of interface elements: $(1) - (5)$ with one level set function and $(6) - (9)$ with two level set functions

Recall that a Delaunay triangulation for a set points P in a plane is a triangulation of the convex set of P such that no point in P is inside the circumcircle of any triangle in this triangulation [40, 77].

Our 2D interface-fitted mesh generation algorithm is described as follows:

Algorithm 1 2D Interface-fitted Mesh Generation Algorithm

```
Input: Grid size, $h$, level set function, $\phi(x)$ and square domain, $\Omega$;
Output: An interface-fitted mesh of $\Omega$;
1. Find all the interface points.
2. Construct a Delaunay triangulation of these points.
3. Remove triangles not in the interface elements and merge all uncut elements.
```

2.1.2 Examples

We give two examples to explain Algorithm 1 in detail. The first example shows the simple case when the interface is a circle. The second example illustrates a more complex case when the interface is unconnected and some interface elements are divided into three parts.

EXAMPLE 2.1 (A circle)**.** Consider the domain $\Omega = (-1, 1)^2$ and a circle interface Γ represented by the level set function $\phi(x, y) = x^2 + y^2 - r^2$, with $r = 0.5$. The interface elements are shown in Fig 2.2.

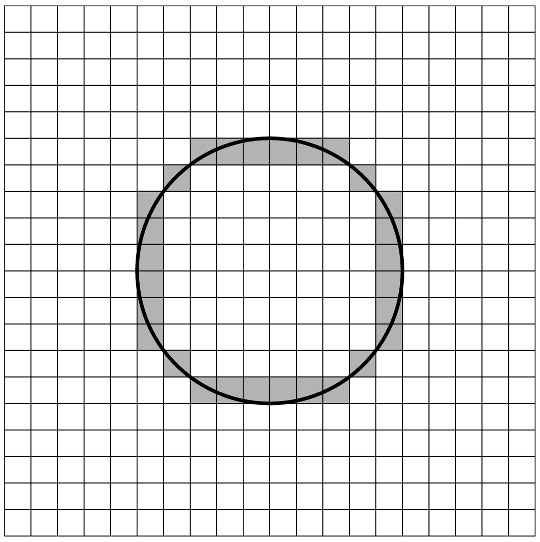
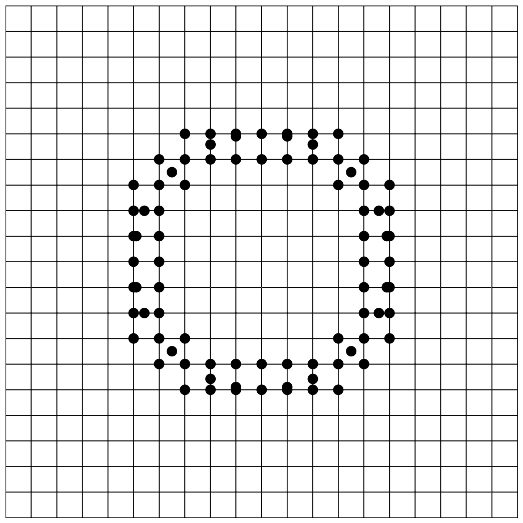


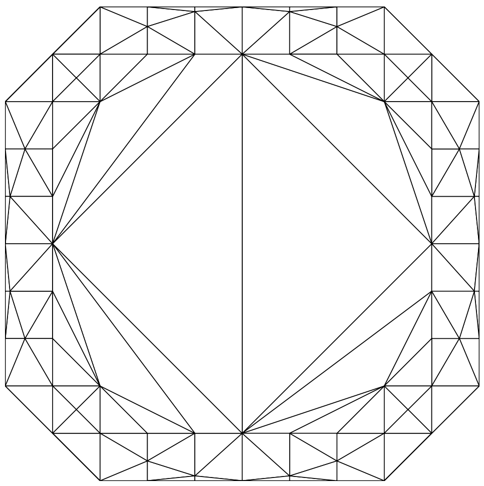
Figure 2.2: Cartesian mesh Ω and a circle interface Γ. The grayed elements are interface elements.

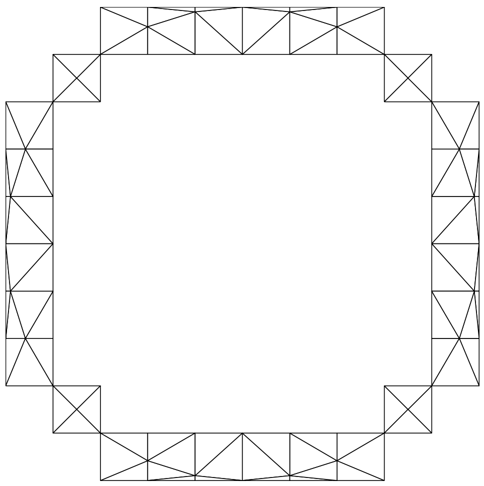
First, we construct a point set $\mathcal{P}$ which includes the intersection points between cut edges and Γ, the vertices of all interface elements, and some auxiliary points. See Fig. 2.3(a) for the illustration. Here we use the bisection method to compute the intersection points within the machine precision tolerance.

Then we construct a Delaunay triangulation based on the point set $\mathcal{P}$. In MATLAB, we just call `DT = delaunay(x,y)` (see Fig. 2.3(b)).

In the last step, we keep the triangles in interface elements and merge the uncut elements to get the final interface-fitted semi-unstructured mesh in Fig. 2.3(c)-(d).


(a) Step 1: Find all the interface points.


(b) Step 2: a Delaunay triangulation of interface points.


(c) Step 3: Remove triangles not in the interface elements.

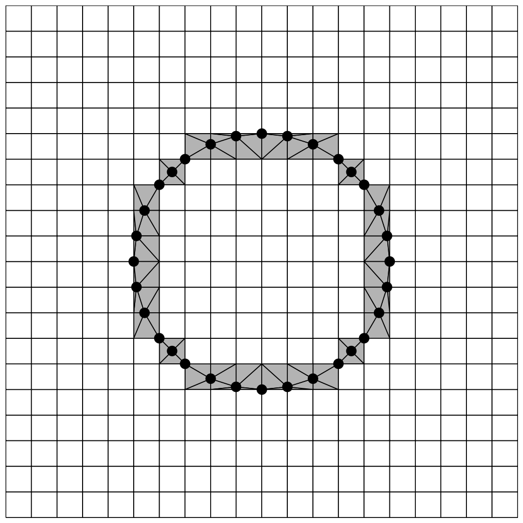

(d) Step 3: Merge all uncut elements to get an interface-fitted mesh.

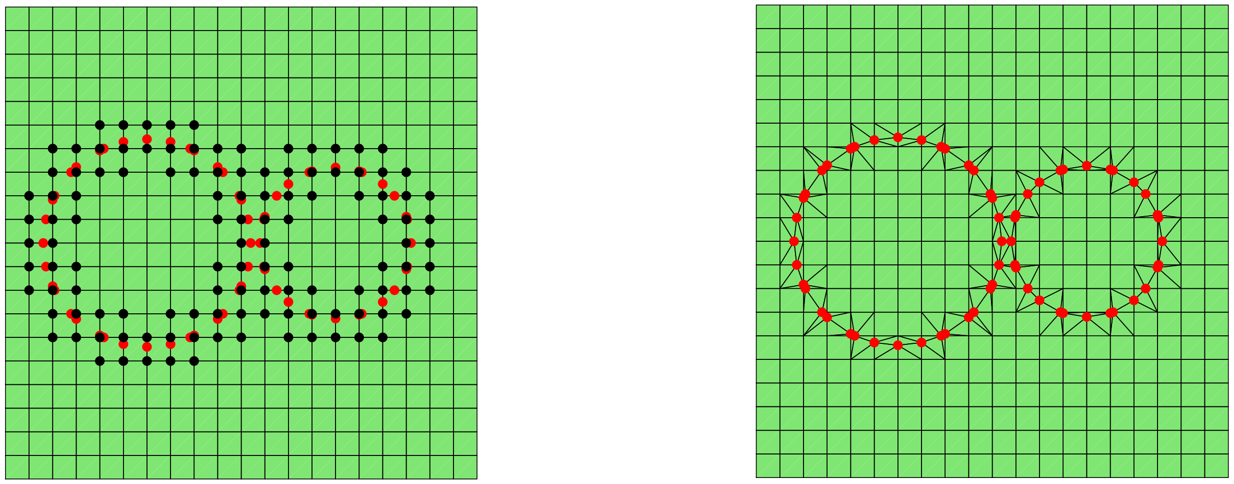
Figure 2.3: Three steps to generate an interface-fitted mesh.

EXAMPLE 2.2 (Two circles)**.** Consider the domain $\Omega = (-1, 1)^2$ and the unconnected interface Γ

represented by the level set function

$$\phi(x,y) = \min\left\{(x+r)^2 + y^2 - (1.1r)^2,\ (x-r)^2 + y^2 - (0.8r)^2\right\},$$

with $r = 0.4$. We can apply the same algorithm and obtain the mesh in Fig 2.4. The only difference is when computing intersection points, we compute them for each level set function separately. We use this example to show our algorithm can handle unconnected interfaces.



(a) Interface points.

(b) The interface-fitted mesh when the interface is two disjoint circles.

Figure 2.4: Interface points and interface-fitted meshes when the interface is unconnected.

2.1.3 Properties

We explore properties of the mesh obtained in Algorithm 1. A triangle is called an interior element when the center point of the triangle is inside. The interface Γ could be approximated by the boundary of those interior elements and can be extracted easily. The obtained discrete interface is denoted by Γ_h.

Proposition 2.1. *The interface will be approximately recovered in the triangulation generated by Algorithm 1. More precisely, we have* $\operatorname{dist}(\Gamma_h, \Gamma) \lesssim h^2$ *provided* Γ *is smooth enough and* h *is* *small enough.*

Proof. We shall use another characterization of Delaunay triangulations: a Delaunay triangulation is the projection of the lower convex hull of points lifted to the paraboloid $f(\vec{x}) = \|\vec{x}\|^2$ [18, 28, 42].

The function values of $f(\vec{x})$ on the four vertices of a square will be on a plane. As the function f is strictly convex, the function value of any intersection points which are different from the vertices of the square will be below this plane. Then the lower convex hull when lifted to $\mathbb{R}^3$ will always connect the intersection points. Thus, the interface will be recovered under this circumstance.

If there are two diagonal vertices of a square on the interface element in P (see Fig. 2.5 (c)), then the Delaunay triangulation on this square is not unique. Using either diagonal of the square is a valid Delaunay triangulation (see Fig. 2.5(a) and (b)). Therefore, we introduce the center of this square as an auxiliary point to make sure the interface is preserved (see Fig. 2.5 (d)).

In both cases, Γ_h contains a piecewise affine approximation of Γ with nodes on the interface and thus the distance is in the order of Ch^2 with constant C depends on the curvature of Γ. □

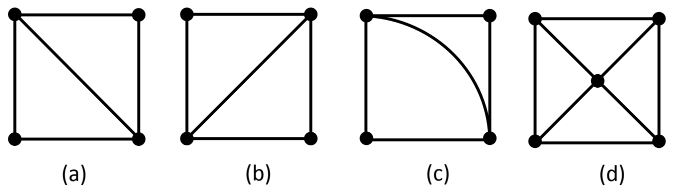


Figure 2.5: Add one auxiliary point when two intersection points are diagonal.

Proposition 2.2. *Assume the mesh size h is small enough such that the interior of each edge has at most one intersection point. Then the maximal angle of the triangulation generated by Algorithm 1 is bounded by* $135°$.

Proof. Let $\mathcal{C}$ be a square with vertices A, B, C, D which intersects with the interface, $\mathcal{S}$ the points set including the vertices of $\mathcal{C}$ and the intersection points, and DT the Delaunay triangulation of

$\mathcal{S}$.

The vertex of every angle in DT can be a vertex of the square or an intersection point. The angle at a square vertex must be bounded by $90°$ as the two rays of the angle is inside the square. Next, let us prove that the angle at an intersection point must be bounded by $135°$. Let E be an intersection point on edge AB, F and G are the other two points of angle $\angle FEG$ and G is on the right of F (see Fig. 2.6). Here F or G can be an intersection point or a vertex of the square.

By our assumption, F or G cannot be in the interior of edge AB and F and G cannot be in the interior of edge CD simultaneously. So either F is on the edge AD or G is on BC. Without loss of generality, we assume G is on BC. Then the angle $\angle FCG \geq 45°$ since F is on the left of the diagonal AC. Note that the triangle $\triangle FCG$ may not be in the DT. Nevertheless, if $\angle FEG > 135°$, then $\angle FEG + \angle FCG > 180°$ which means the circumcircle of ΔFEG must include vertices C violating the Delaunay property. So $\angle FEG$ must be bounded by $135°$. □

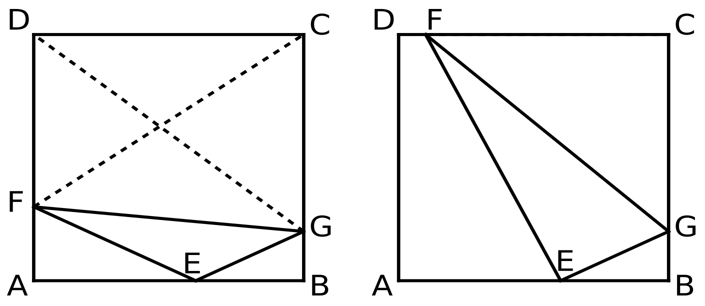


Figure 2.6: The angle $\angle FEG$ at the intersection point E.

Let N be the number of nodes. Since we restrict the Delaunay triangulation on a local region near the interface, the complexity of generating a Delaunay triangulation will be $\mathcal{O}(N^{1/2}\log N)$ in 2D which can be ignored compared with the $\mathcal{O}(N)$ complexity of assembling the matrix and solving the matrix equation. Such localization will make it possible to track the moving interface, which will be discussed in Chapter 4.

The overall complexity of our mesh generation algorithm is: $c_1 N + c_2 N^{1/2}\log N$ since we need to compute the sign of the level set function at N vertices. In practice, however, the constant $c_1 \ll c_2$

and the time scales like $\mathcal{O}(N^{1/2})$.

2.2 Interface-fitted Mesh Generator: Three Dimensions

In this section, we present a novel mesh generation algorithm to generate an interface-fitted mesh for a given smooth interface in three dimensions. We begin with a brief review on the difficulty of 3D mesh generation and then introduce our algorithm to overcome this difficulty.

2.2.1 Main Difficulty

Tetrahedral meshes are frequently used in classical finite element methods. The size and shape of the tetrahedra influence the accuracy of finite element solutions [118]. The quality of the tetrahedron's shape can be measured by using the aspect ratio or the radius-edge ratio. The aspect ratio of a tetrahedron is usually defined as its circumradius divided by its inradius and the radius-edge ratio is the circumradius divided by the shortest edge length of the tetrahedron. The aspect ratio or radius-edge ratio of a mesh is the largest corresponding ratio of all of its tetrahedral elements. If the aspect ratio or radius-edge ratio of a mesh are small, we called the mesh well-shaped [40, 81]. Ideally we expect each element in the mesh is shape regular. But violation is allowed as long as the so-called maximal angle condition is satisfied [1, 6, 17, 38, 75].

The difficulty of mesh generation in three dimensions is due to the existence of slivers. Slivers have small radius-edge ratio, but large aspect ratio, which are considered as bad-shaped tetrahedral elements. The results of the accuracy and convergence of finite element methods may not hold anymore in the existence of slivers which violates the maximal angle condition. A lot of methods have been developed to remove slivers; see e.g. [30, 31, 41]. Sliver removal methods, however, involve the addition and rearrangement of points and thus destroy the semi-structure of the mesh.

We shall introduce polyhedral meshes near the interface to remove slivers. When we get the interface elements (which is defined similarly to 2D and will be made clear later) and intersection points (the definition is the same as in 2D), we can divide interface elements into polyhedra instead of tetrahedra. Slivers will be eliminated and part of their faces will become the faces of polyhedral elements. For example, when the interface cuts across one element with four almost coplanar intersection points, if we divide the element into tetrahedra, then the four intersection points could form a sliver (see Fig. 2.7). If we use a polyhedral mesh, however, the two well-shaped triangles will become the boundary of two polyhedra.

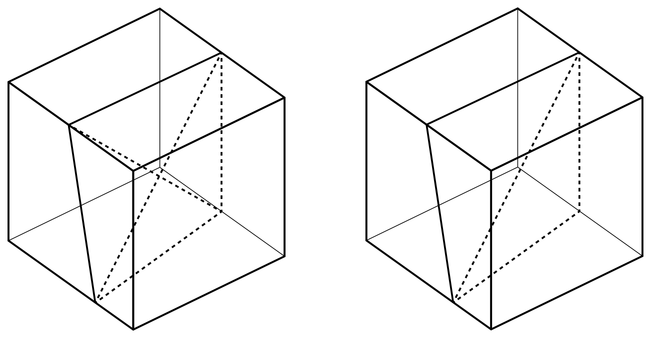


Figure 2.7: Sliver exists (left) and is removed when the element is divided into polyhedra (right).

2.2.2 Algorithm

We write down the algorithm and then explain the details step by step.

Algorithm 2 3D Interface-fitted Mesh Generation Algorithm

```
Input: Grid size $h$, level set function $\phi(x)$, and a cubic domain $\Omega$;
Output: Interface-fitted mesh $\Omega$;
1. Find all the interface points.
2. Construct the Delaunay mesh $DT$ on these points.
3. Post processing: remove unnecessary tetrahedra in $DT$, merge tetrahedra into polyhedra, and merge with uncut elements.
```

Given a cubic domain Ω which includes the interface Γ described as the zero level set of ϕ, and a mesh size h, we first generate the uniform Cartesian mesh of Ω with size h. The cubes in the

Cartesian mesh in domain Ω could be classified into exterior, interior and interface elements by checking the sign of the centers of the cubes. We label them by 1, -1 and 0 respectively.

We define interface elements as elements satisfying one of the following rules:

1. There exists at least two vertices p and q with opposite sign, namely $\phi(p)\phi(q) < 0$;
2. There exists at least three vertices on the interface.

All interface elements will form a hexahedral mesh of a layer of the interface (see Fig. 2.8). All boundary faces of this hexahedral mesh are extracted and will be used as faces of the polyhedral mesh for the interface. Note that these boundary faces are square faces.

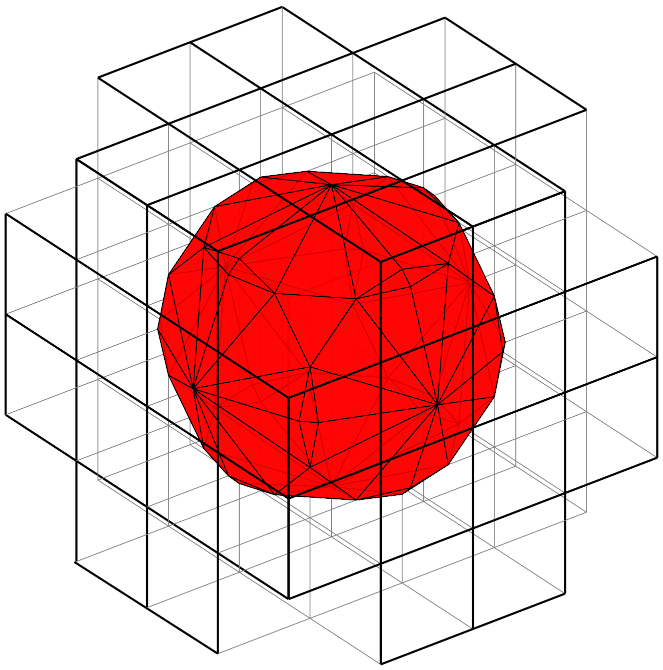


Figure 2.8: The surface of the interface is embedded in the hexahedron.

In step 1, similarly to two dimensions, we find cut edges and intersection points, and add auxiliary points if necessary. The criterion of adding auxiliary points is the same as 2D: if a square face contains two opposite vertices on the interface, we will add the center point of this square face as the auxiliary point.

In step 2, we generate a Delaunay mesh DT of $\mathcal{P}$, the set of interface points, whose definition is the same as that in two dimensions.

In step 3, we post-process DT to get a polyhedral mesh near the interface and merge all uncut cubic elements away from the interface.

Similar to the 2D case, we only keep tetrahedra inside the interface elements, which might contain slivers, and remove tetrahedra not in the interface elements which can be easily marked by checking the center of tetrahedra in DT.

Now we have a tetrahedral mesh of all the interface elements, and we still call this tetrahedral mesh as DT for convenience. We could split the tetrahedron in DT into two categories: exterior tetrahedral set DT_E and interior tetrahedral set DT_I. For a tetrahedron in DT, if the minimum of the sign function of the ϕ value of the four vertex nodes is -1, we put it into DT_I, otherwise, we add it into DT_E. The interface Γ could be extracted using the boundary faces of DT_I and the normal direction of the extracted surface mesh points outside of the interface. Tetrahedra in each category will be merged into polyhedra element by element.

Instead of storing all vertices of a polyhedron, we shall store the polyhedral mesh by the data structure `face` and `face2elem`. The array `face` records indices of three (triangular face) or four (square face) vertices of all faces. The direction of all faces follows the right-hand side rule, that is, the normal direction of each face is outwards. The array `face2elem` records the index of the polyhedron to which the faces belong.

Fig. 2.9 is a simple example. Given a unit cube with three intersection points, it is divided into two polyhedra. Each polyhedron is stored by faces and elements to which they belong. The values of `face` and `face2elem` in Fig. 2.9 are shown in Table 2.1.

Notice that some face, e.g., `[3 7 8 4]` is stored as a square instead of two triangles since this face is shared by another cube which is not included as an interface element. Those square faces are

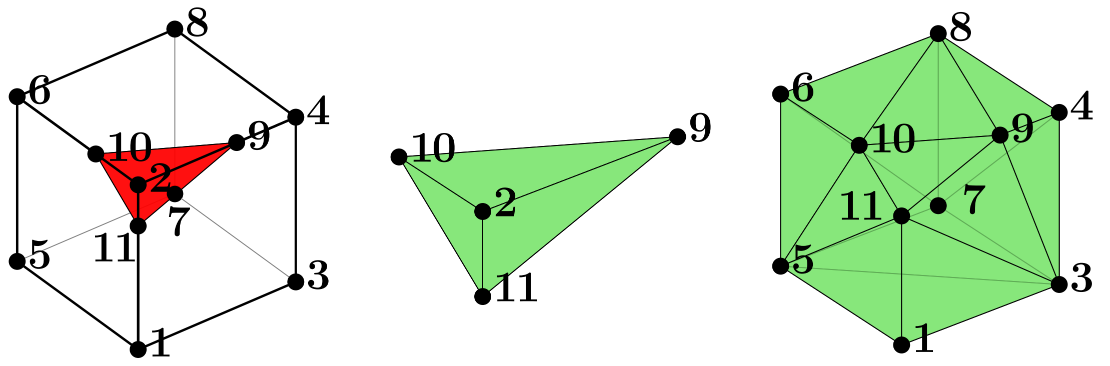


Figure 2.9: An interface element is divided into two polyhedra.

boundary faces of the hexahedral mesh which consists of all interface elements. All such square faces have been extracted when we collect all interface elements.

Every interface element is divided into several polyhedra according to our method. For each polyhedron, we need to assign a unique index. This index map from face to element, `face2elem`, can be generated in two stages. In most cases, the interface element is just divided into two polyhedra. In the first stage, for the interior part, we use the original interface element index j and for the exterior part, we append a new index $j + N$, where N is the number of elements in the initial Cartesian grid. In some cases, however, one cube could be divided into three or more polyhedra (see the three cases in Fig. 2.10). In the second stage, we use Euler's formula to check the connectedness of the obtained polyhedral mesh. If a disconnected polyhedron is found, we group faces into different connected components which is equivalent to dividing the original polyhedron into more polyhedra. Thanks to our data structure, we only need to change `face2elem` when adding and storing the new polyhedra.

In a nutshell, we could get a polyhedral mesh near the interface by storing the triangular and square faces. The final interface-fitted mesh consists of polyhedra near the interface and uncut (cube) elements away from the interface.

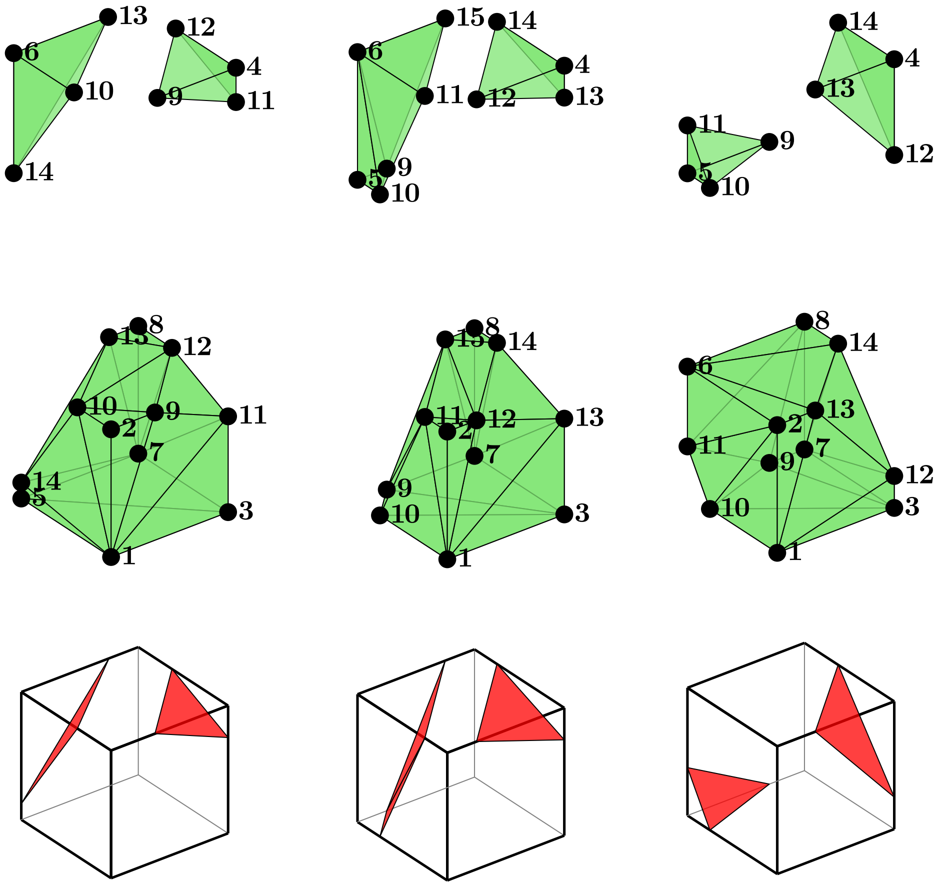


Figure 2.10: An cube is divided into three parts.

	1	2	3	4
1	2	9	10	
2	2	11	9	
3	2	10	11	
4	11	10	9	
5	11	1	3	
6	5	1	11	
7	3	4	9	
8	3	9	11	
9	9	4	8	
10	5	10	6	
11	5	11	10	
12	10	8	6	
13	10	9	8	
14	11	9	10	
15	1	5	7	3
16	5	6	8	7
17	3	7	8	4

	1
1	1
2	1
3	1
4	1
5	2
6	2
7	2
8	2
9	2
10	2
11	2
12	2
13	2
14	2
15	2
16	2
17	2

Table 2.1: The `face` array (left) and `face2elem` (right) for two polyhedra in Fig. 2.9.

2.2.3 Properties

The generated Delaunay triangulation will recover the interface by the lifting method. Namely Proposition 2.1 also holds for the 3D case since the characterization of a Delaunay triangulation as the projection of the lower convex hull holds in general dimensions. We formally summarize below.

Proposition 2.3. *The interface will be approximately recovered in the triangulation generated by Algorithm 2. More precisely, we have* $\operatorname{dist}(\Gamma_h, \Gamma) \lesssim h^2$ *provided* Γ *is smooth enough and* h *is small enough.*

Next we shall show the maximum angle of the surface mesh is uniformly bounded by $144°$. In [109], the author considers 12 types of subdivision of boundary cells (not necessarily satisfying the Delaunay property) in three dimensions and shows the same bound.

Proposition 2.4. *The maximal angle of the triangular faces of the polyhedral mesh is bounded by*

144°.

Proof. For simplicity, let $\mathcal{C}$ be a unit cube which intersects with the interface, and $\mathcal{S}$ the set of points including the eight cube vertices and the intersection points. Let DT be the 3D Delaunay triangulation on $\mathcal{S}$.

For a 3D Delaunay triangulation, it satisfies the Delaunay empty sphere property such that no point in $\mathcal{S}$ is inside the circumsphere of any tetrahedron of DT. Given a tetrahedron T in DT which has a triangular face τ on one (denoted as $\mathcal{F}$) of the six square faces of $\mathcal{C}$. Since, on the plane spanned by $\mathcal{F}$, the circumcircle of τ is also on the circumsphere of T, then by Delaunay empty sphere property, there is no point of S on $\mathcal{F}$ which is inside the circumcircle of τ, namely, the boundary triangulation of DT on $\mathcal{F}$ is also Delaunay, and thus the maximal angle of these triangles is bounded by 135° by Proposition 2.2.

Next we only need to consider the interface triangles with three vertices on the interface. For these interface triangles, their angles can be divided into 16 cases (see Fig. 2.11).

In case (1) to (15), one can find the upper bound of the angle by calculus analysis. Here we take the case (1) as an example to show how to find the upper bound, see Fig. 2.11 (1). Let v_{LA} be the vector from point L to point A and $|v_{LA}|$ the length of v_{LA}. Similarly, we have vectors $v_{LH}, v_{LN}, v_{NA}, v_{NH}$, then

$$\begin{aligned}\cos\angle ALH =& \frac{v_{LA}\cdot v_{LH}}{|v_{LA}||v_{LH}|} = \frac{(v_{LN}+v_{NA})\cdot(v_{LN}+v_{NH})}{\sqrt{|v_{LN}|^2+|v_{NA}|^2}\sqrt{|v_{LN}|^2+|v_{NH}|^2}}\\ =& \frac{|v_{LN}|^2}{\sqrt{|v_{LN}|^2+|v_{NA}|^2}\sqrt{|v_{LN}|^2+|v_{NH}|^2}} \geq 0.\end{aligned}$$

When $|v_{LN}| = 0$, $\cos\angle ALH$ reaches the minimum value zero, namely, the maximum of $\angle ALH$ is 90°. By the similar method, one can get the upper bounds for other cases except case (16) in Fig. 2.11.

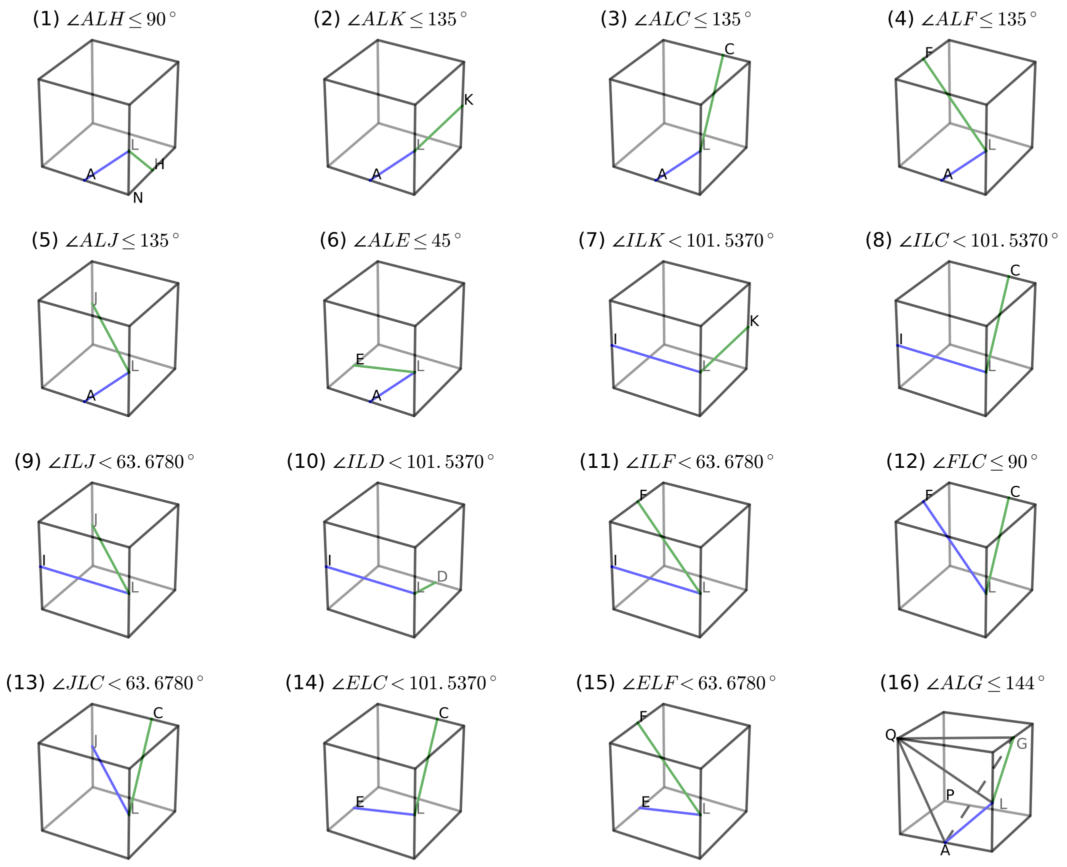

(1) $\angle ALH \leq 90^\circ$
(2) $\angle ALK \leq 135^\circ$
(3) $\angle ALC \leq 135^\circ$
(4) $\angle ALF \leq 135^\circ$
(5) $\angle ALJ \leq 135^\circ$
(6) $\angle ALE \leq 45^\circ$
(7) $\angle ILK < 101.5370^\circ$
(8) $\angle ILC < 101.5370^\circ$
(9) $\angle ILJ < 63.6780^\circ$
(10) $\angle ILD < 101.5370^\circ$
(11) $\angle ILF < 63.6780^\circ$
(12) $\angle FLC \leq 90^\circ$
(13) $\angle JLC < 63.6780^\circ$
(14) $\angle ELC < 101.5370^\circ$
(15) $\angle ELF < 63.6780^\circ$
(16) $\angle ALG \leq 144^\circ$
A
L
H
N
K
C
F
J
E
I
D
Q
G
P

Figure 2.11: Different angle cases in the interface triangles.

In case (16), provided ΔALG is an interface triangle and $\angle ALG$ is the angle bigger than $144°$. By Algorithm 2, there must exist a vertex of $\mathcal{C}$, for example, vertex Q and $ALGQ$ is a tetrahedron in the Delaunay triangulation. Let $(1-h_1, 0, 0), (1, 0, h_2)$ and $(1, h_3, 1)$ be the coordinates of A, L and H, respectively. Then one can get the circumcenter O and circumradius r of the circumsphere of $ALGQ$, then construct function $f(h_1, h_2, h_3) := r - |P - O|$. By the assumption $\angle ALG > 144°$ and the 2D Delaunay empty circle property on the boundary face of $\mathcal{C}$, one can show that $f(h_1, h_2, h_3) > 0$, namely P is inside of the circumsphere of tetrahedron $ALGQ$, which contradicts with the Delaunay empty sphere property. □

Remark 2.5. For the proof of the 3D angle case (16), we use the *region_plot* function in SageMath [36] to show $f(h_1, h_2, h_3) > 0$ under the given assumptions.

Remark 2.6. We emphasize that the 16 cases plotted above are used to prove the maximal angle condition. In the algorithm, we get the mesh by directly calling Delaunay algorithm with all interface points as input.

Again, we restrict the Delaunay triangulations on a local region near the interface. The overall complexity of our mesh generation algorithm is: $c_1 N + c_2 N^{2/3} \log N$ since we need to compute the sign of the level set function at N vertices with $c_1 \ll c_2$. The meshing time scales like $\mathcal{O}(N^{2/3})$. See Section 3.4 for numerical results.

In summary, our mesh generator is simple and fast. The generated mesh is semi-unstructured. The interface is approximately recovered, and the maximum angle of the surface mesh is uniformly bounded.

Chapter 3

Finite Element Methods for Elliptic Interface Problems

We start with Sobolev spaces and the weak formulation of the elliptic interface problem (1.1)-(1.4). We then introduce the linear virtual element methods and discuss the implementation detail. Finally we present several numerical results within the 3D setting for elliptic interface problems with jump conditions across the interface.

3.1 Sobolev Spaces and Weak Formulation

Let D denote a bounded and open set in $\mathbb{R}^d, d = 2, 3$ and $W^{m,p}(D)$ be the usual Sobolev space with standard norm $\|\cdot\|_{m,p,D}$ and semi-norm $|\cdot|_{m,p,D}$. In particular, for $p = 2$, we denote $H^m(D) = W^{m,p}(D)$ and the corresponding norm and semi-norm by $\|\cdot\|_{m,D} = \|\cdot\|_{m,p,D}$ and $|\cdot|_{m,D} = |\cdot|_{m,p,D}$, respectively. The space $H_0^1(D) = \{v \in H^1(D) : v|_{\partial D} = 0\}$ is the subspace of $H^1(D)$ with zero trace. Let $(\cdot,\cdot)_D$ and $\langle\cdot,\cdot\rangle_{\partial D}$ denote the standard L^2 inner products of $L^2(D)$ and $L^2(\partial D)$ respectively.

Domains are considered as open sets. Define $\tilde{\Omega} = \Omega^- \cup \Omega^+$ and notice that $\Omega = \Omega^- \cup \Gamma \cup \Omega^+ = \tilde{\Omega} \cup \Gamma$. For $v \in W^{m,p}(\tilde{\Omega})$, that is, $v|_{\Omega^-} \in W^{m,p}(\Omega^-)$ and $v|_{\Omega^+} \in W^{m,p}(\Omega^+)$, v may not be in $W^{m,p}(\Omega)$ due to the jump across the interface Γ.

To derive the weak formulation of elliptic interface problems (1.1)-(1.4), we multiply (1.1) with a test function $v \in H_0^1(\Omega)$ and apply integration by parts. To address the jump of function values, we chose a $w^- \in H^1(\Omega^-)$ with $w^- = q_0$ on $\partial\Omega^-$. With a slight abuse of notation, the zero extension of w^- to $H^1(\tilde{\Omega})$ is still denoted by w^-. The model (1.1)-(1.4) is equivalent to: find $p \in H_g^1(\Omega) = \{v \in H^1(\Omega) : v|_{\partial\Omega} = g\}$ such that

$$(\beta\nabla p, \nabla v)_\Omega = (f, v)_\Omega - \langle q_1, v\rangle_\Gamma + (\beta\nabla w^-, \nabla v)_{\Omega^-}, \quad \forall v \in H_0^1(\Omega), \tag{3.1}$$

and set $u = p - w^-$. It is easy to show that u solves equations (1.1)-(1.4). Even though the choice of w^- is not unique, the solution u does not depend on the choice of w^- by the maximal principle. The flux jump $[\beta u_n]_\Gamma = q_1$ is imposed in $H^{-1/2}(\Gamma)$ and the jump of function value is imposed in $H^{1/2}(\Gamma)$.

3.2 Finite Element Methods in 2D

In this section, we present the numerical analysis of the standard finite element methods on the two-dimension interface-fitted mesh generated by the Algorithm 1.

For simplicity of exposition, we assume the function value jump condition $[u]_\Gamma = 0$. Let $\mathcal{T}_h$ be an interface-fitted triangular mesh with maximal angles uniformly bounded away from π. For each $\tau \in \mathcal{T}_h$, let h_τ denote its diameter and $h = \max_{\tau\in\mathcal{T}_h} h_\tau$. The vertices on Γ forms a polygon Γ_h approximation of Γ. The polygon also splits Ω into two subdomains, Ω_h^+ and Ω_h^-, which are the approximations of Ω^+ and Ω^-, respectively. Each triangle $\tau \in \mathcal{T}_h$ is in either Ω_h^+ or Ω_h^- and has at most two vertices on Γ.

Let V_h be the linear finite element space on $\mathcal{T}_h$. The linear finite element approximation of (3.1) is as follows: find $u_h \in V_h \cap H_0^1(\Omega)$ such that:

$$(\beta_h \nabla u_h, \nabla v_h)_\Omega = (f, v_h)_\Omega - \langle \bar{q}_1, v \rangle_{\Gamma_h}, \quad \forall v_h \in V_h \cap H_0^1(\Omega), \tag{3.2}$$

where $\bar{q}_1 = q_1(\mathcal{P}_0(x))$ and $\mathcal{P}_0(x)$ is a well defined projection from Γ_h to Γ (c.f. [129]).

We can get the nearly optimal L^2-norm and H^1-norm estimates as the results in [136, 29].

THEOREM 3.1. *Let u be the solution of* (3.1) *and u_h be the linear finite element approximation in* (3.2) *based on the two-dimension interface-fitted mesh generated by the Algorithm 1. We have*

$$\|\beta^{1/2}(\nabla u - \nabla u_h)\|_{0,\Omega} \lesssim h|\log h|^{1/2}(\|f\|_{0,\Omega} + \|q_1\|_{2,\Gamma}), \tag{3.3}$$

$$\|u - u_h\|_{0,\Omega} \lesssim h^2|\log h|(\|f\|_{0,\Omega} + \|q_1\|_{2,\Gamma}). \tag{3.4}$$

Proof. For finite element approximation, we have the Ceá lemma,

$$\|\beta^{1/2}(\nabla u - \nabla u_h)\|_{0,\Omega} \leq \|\beta^{1/2}(\nabla u - \nabla u_I)\|_{0,\Omega}.$$

Then the energy error estimate is reduced to the interpolation error estimate. In [6], the authors proved that the local interpolation error estimate $\|(\nabla u - \nabla u_I)\|_{0,\tau} \lesssim h\|u\|_{2,\tau}$ provided the maximal angle condition is satisfied which has been verified for the interface-fitted mesh generated by Algorithm 1; see Proposition 2.2. Another difficulty is the mis-match of the curved interface and the discrete interface. Then follow the proof in [136, 29], and replace the mesh regular condition there by the maximal condition, we obtain the desired results. □

A mesh is $\mathcal{O}(h^{2\sigma})$ irregular means the total area of all adjacent triangle pairs in $\mathcal{T}_h$ which do not form an $\mathcal{O}(h^2)$ approximate parallelogram is $\mathcal{O}(h^{2\sigma})$. For the interface-fitted mesh generated by

Algorithm 1, only the adjacent triangle pairs near the interface is not $\mathcal{O}(h^2)$ approximate parallelogram and other adjacent triangle pairs away from the interface can exactly form a parallelogram. That is $\sigma = 0.5$ for the mesh generated by Algorithm 1.

Then follow the proof procedure in [129], we can also prove the following superconvergence result.

THEOREM 3.2. *If $u \in H^1(\Omega) \cap H^3(\tilde{\Omega}) \cap W^{2,\infty}$ and Γ is of class $\mathcal{C}^2$, then for all $v_h \in V_h$,*

$$\|\beta_h^{1/2}(\nabla u_h - \nabla u_I)\|_{0,\Omega} \lesssim h^{3/2} \left(\|u\|_{3,\tilde{\Omega}} + \|u\|_{2,\infty,\tilde{\Omega}} + \|u\|_{2,\infty,\tilde{\Omega}} + \|q_1\|_{0,\infty,\Gamma}\right). \tag{3.5}$$

Let $h_{\min}$ be the minimum element size of $\mathcal{T}_h$, by the discrete embedding result,

$$\|v_h\|_{0,\infty,\Omega} \lesssim |\log h_{\min}|^{1/2} |v_h|_{1,\Omega}, \text{ for all } v_h \in V_h \cap H_0^1(\Omega), \tag{3.6}$$

we have the error estimate for the maximal norm estimate.

Corollary 3.3. *Assume the same hypothesis in Theorem 3.2. Then*

$$\begin{aligned}\|\beta_h^{1/2}(\nabla u_h - \nabla u_I)\|_{0,\infty,\Omega} \lesssim |\log h_{\min}|^{1/2} \big[h^{3/2}(&\|u\|_{3,\tilde{\Omega}} + \|u\|_{2,\infty,\tilde{\Omega}} \\ &+\|u\|_{2,\infty,\tilde{\Omega}} + \|q_1\|_{0,\infty,\Gamma})\big]\end{aligned}$$

3.3 Virtual Element Methods in 3D

In this section, we focus on solving three-dimensional elliptic equations by the virtual element methods (VEM) developed by Brezzi's group [10, 11].

Let $\mathcal{T}_h$ be the interface-fitted polyhedral mesh generated by the algorithm in Section 2.2. Recall that elements near the interface Γ are polyhedra with triangular or square faces and a uniform cubic mesh away from the interface. We could not use the classical finite element methods which are not well-defined on polyhedra. Instead, we shall apply virtual element methods [10] which can be thought of as conforming finite element spaces defined on polyhedral meshes.

A local finite-dimensional vector space $V_h(E)$ for a polyhedron $E \in \mathcal{T}_h$ is defined as

$$
\begin{aligned}
V_h(E) := \{v \in H^1(E) : \Delta v|_E = 0, v|_{\partial E} \text{ is continuous and} \\
\text{piecewise linear (on triangles) or bilinear (on squares)}\}.
\end{aligned}
$$

As a piecewise linear or bilinear function will be uniquely determined by its value on vertices, $\dim V_h(E) = n_E^v$, where n_E^v is the number of vertices of E.

We define the global virtual element space

$$
V_h = \{v_h \in H^1(\Omega) : v_h|_E \in V_h(E) \text{ for all } E \in \Omega_h\}.
$$

Let $\mathcal{N}(\mathcal{T}_h)$ be the set of vertices of mesh $\mathcal{T}_h$ and $N = |\mathcal{N}(\mathcal{T}_h)|$ be the number of vertices. We define the operator dof_i from V_h to $\mathbb{R}$ as $\mathrm{dof}_i(v_h) = v_h(\boldsymbol{x}_i)$, for a vertex $\boldsymbol{x}_i \in \mathcal{N}(\mathcal{T}_h)$. The canonical basis $\{\phi_1, \cdots, \phi_N\} \subset V_h$ is chosen as $\mathrm{dof}_i(\phi_j) = \delta_{ij}, i, j = 1, \cdots, N$. And the nodal interpolation $I_h : C(\bar{\Omega}) \to V_h$ is defined as $I_h u = \sum_{i=1}^N u(\boldsymbol{x}_i)\phi_i$ and denoted by $u_I = I_h u$. The basis does not need to be written explicitly which is the main difference between classical finite element methods and virtual element methods.

As mentioned before, we could extract an approximate surface Γ_h which splits Ω into two subdomains: Ω_h^- and Ω_h^+, which are the approximation of Ω^- and Ω^+, respectively. Similarly, $\beta_h|_\tau = \beta^+$ for all $\tau \in \Omega_h^+$ and $\beta_h|_\tau = \beta^-$ for all $\tau \in \Omega_h^-$.

Let w_I^- be the nodal interpolation of w^- in V_h. A simple construction is one that: interpolates q_0 on Γ_h and sets other coefficients to zero. The linear virtual element approximation of (3.1) is: finding

$p_h \in V_h \cap H^1_g(\Omega)$ such that:

$$(\beta_h \nabla p_h, \nabla v_h)_\Omega = (f, v_h)_\Omega - \langle q_1, v_h\rangle_\Gamma + (\beta_h \nabla w_h^-, \nabla v_h)_{\Omega^-}, \quad \forall v_h \in V_h \cap H^1_0(\Omega)$$

and taking $u_h = p_h - w_h^-$. Suppose $p_h = \sum_{j=1}^N p_j \phi_j$, $w_h^- = \sum_{j=1}^N w_j \phi_j$, by linearity, we have for $i \in 1, \cdots, N$,

$$\sum_{j=1}^N (\beta_h \nabla \phi_j, \nabla \phi_i)_\Omega p_j = (f, \phi_i)_\Omega - \langle q_1, \phi_i\rangle_\Gamma + \sum_{j=1}^N (\beta_h \nabla \phi_j, \nabla \phi_i)_{\Omega^-} w_j. \tag{3.7}$$

We define the matrix $(\boldsymbol{A}_h^-)_{ij} = (\beta_h^- \nabla \phi_j, \nabla \phi_i)_{\Omega_h^-}$, $(\boldsymbol{A}_h^+)_{ij} = (\beta_h^+ \nabla \phi_j, \nabla \phi_i)_{\Omega_h^+}$ and $(\boldsymbol{A}_h)_{ij} = (\beta_h \nabla \phi_j, \nabla \phi_i)_{\Omega_h}$ in Ω_h. Then $\boldsymbol{A}_h = \boldsymbol{A}_h^- + \boldsymbol{A}_h^+$. Define the vector $\boldsymbol{b} = (b_1, \cdots, b_N)^t$ by $b_i = (f, \phi_i)_\Omega - \langle q_1, \phi_i\rangle_\Gamma$. Equation (3.7) is written in the matrix form as

$$\boldsymbol{A}_h \boldsymbol{p}_h = \boldsymbol{b} + \boldsymbol{A}_h^- \boldsymbol{w}_h, \tag{3.8}$$

where $\boldsymbol{A}_h$ and $\boldsymbol{A}_h^-$ are $N \times N$ matrices, $\boldsymbol{p}_h = (p_h^1, \cdots, p_h^N)^t$ and $\boldsymbol{w}_h = (w_1, \cdots, w_N)^t$. Since the coefficient β is a positive constant, the matrices $\boldsymbol{A}_h$ and $\boldsymbol{A}_h^-$ are symmetric and positive definite. The algebraic system (3.8) could be solved stably and efficiently by using algebraic multigrid methods.

For finite element methods, it suffices to compute the local stiffness matrix in each element and then, based on that, the matrices $\boldsymbol{A}_h^+, \boldsymbol{A}_h^-$ are assembled by summing the contribution from each element. Therefore, the major task is to compute $(\nabla \phi_j, \nabla \phi_i)_E$.

To do so, we introduce some projection operators at first. For each polyhedron E, the operator $\Pi^\nabla : V_h(E) \to \mathbb{P}_1(E)$ is defined as the H^1 projection to $\mathbb{P}_1(E)$ space, i.e.,:

$$(\nabla p_k, \nabla \Pi^\nabla v_h)_E = (\nabla p_k, \nabla v_h)_E \quad \text{for all } p_k \in \mathbb{P}_1(E),$$

where $\mathbb{P}_1(E)$ is the space of linear polynomials. As it can be easily seen that the above condition defines $\Pi^\nabla v_h$ only up to a constant. This can be fixed by prescribing a projection operator onto constants $P_0 : V_h(E) \to \mathbb{P}_0(E)$ and requiring

$$P_0(\Pi^\nabla v_h - v_h) = 0.$$

One such choice is $P_0 v_h = \sum_{i=1}^{n_E^v} v_h(\boldsymbol{x}_i)/n_E^v = \sum_{i=1}^{n_E^v} \mathrm{dof}_i(v_h)/n_E^v$.

Using the projection Π^∇, we write the basis function $\phi_i \in V_h(E)$ as $\Pi^\nabla \phi_i + (I - \Pi^\nabla)\phi_i$ and split the entry of the local stiffness matrix as

$$(\nabla \Pi^\nabla \phi_i, \nabla \Pi^\nabla \phi_j)_E + (\nabla(I - \Pi^\nabla)\phi_i, \nabla(I - \Pi^\nabla)\phi_j)_E.$$

Again the second term is not computable since the basis ϕ_i is not known point-wise. Instead we replace by a so-called stabilization term $S^E(\cdot,\cdot)$

$$(\nabla \Pi^\nabla \phi_i, \nabla \Pi^\nabla \phi_j)_E + S^E((I - \Pi^\nabla)\phi_i, (I - \Pi^\nabla)\phi_j).$$

Let h_E be $|E|^{1/3}$, where $|E|$ means the volume of E. We use a scaled l^2 inner product in the stabilization term

$$S^E((I - \Pi^\nabla)\phi_i, (I - \Pi^\nabla)\phi_j) = h_E \sum_{r=1}^{n_E^v} \mathrm{dof}_r((I - \Pi^\nabla)\phi_i)\, \mathrm{dof}_r((I - \Pi^\nabla)\phi_j)$$

in order to satisfy the assumption of S^E

$$c_1(\nabla v, \nabla v) \le S^E(v, v) \le c_2(\nabla v, \nabla v), \quad \forall v \in V_h(E) \text{ and } \Pi^\nabla v = 0$$

for some positive constants c_1 and c_2 independent of E and h_E. The explicit expression of the local stiffness matrix of the virtual element method is:

$$(\boldsymbol{K}_h^E)_{ij} := (\nabla\Pi^\nabla\phi_i, \nabla\Pi^\nabla\phi_j)_E + h_E\sum_{r=1}^{n_E^v}\mathrm{dof}_r((I-\Pi^\nabla)\phi_i)\,\mathrm{dof}_r((I-\Pi^\nabla)\phi_j).$$

We now give concrete formulae on the computation of the matrix representation of the operator Π^∇. Let $\boldsymbol{x}_E = (x_E, y_E, z_E)$ be the center of E, i.e. $\boldsymbol{x}_E = 1/n_E^v\sum_{i=1}^{n_E^v}\boldsymbol{x}_i$. We choose a scaled monomial basis of $\mathbb{P}_1(E)$ as $m_1 = 1, m_2 = (x-x_E)/h_E, m_3 = (y-y_E)/h_E, m_4 = (z-z_E)/h_E$.

Let $\boldsymbol{G}_{4\times 4}$ be defined as

$$\boldsymbol{G} := \begin{pmatrix} P_0 m_1 & P_0 m_2 & \cdots & P_0 m_4 \\ 0 & (\nabla m_2, \nabla m_2)_{0,E} & \cdots & (\nabla m_4, \nabla m_2)_{0,E} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & (\nabla m_2, \nabla m_4)_{0,E} & \cdots & (\nabla m_4, \nabla m_4)_{0,E} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & h_E\boldsymbol{I}_3 \end{pmatrix}$$

where $\boldsymbol{I}_3$ is a 3×3 identity matrix.

Let $\boldsymbol{B}_{4\times n_E^v}$ be a matrix defined as:

$$\boldsymbol{B} := \begin{pmatrix} P_0\phi_1 & \cdots & P_0\phi_{n_E^v} \\ (\nabla m_2, \nabla\phi_1)_E & \cdots & (\nabla m_2, \nabla\phi_{n_E^v})_E \\ \vdots & \ddots & \vdots \\ (\nabla m_4, \nabla\phi_1)_E & \cdots & (\nabla m_4, \nabla\phi_{n_E^v})_E \end{pmatrix}.$$

The formulae for the first row of $\boldsymbol{B}$ is $P_0\phi_1 = P_0\phi_2 = \ldots = P_0\phi_{n_E^v} = 1/n_E^v$. For the other components $(\nabla m_j, \nabla\phi_i)_E, j = 2, 3, 4$, we have $(\nabla m_j, \nabla\phi_i)_E = -\int_E \Delta m_j\phi_i + \int_{\partial E}\frac{\partial m_j}{\partial n}\phi_i$ by integration by parts. The first term is zero as $\Delta m_j = 0$ for linear polynomials. We only need to compute the second term. Due to our data structure, all the faces on the ∂E are either triangles or squares. Then

$$\int_{\partial E} \frac{\partial m_j}{\partial n} \phi_i = \frac{\sum_{i \in \text{triangular face} f} n_f^j \, |f|}{3h_E} + \frac{\sum_{i \in \text{square face} f} n_f^j \, |f|}{4h_E}, \tag{3.9}$$

where $\boldsymbol{n_f} = (n_f^x, n_f^y, n_f^z) = (n_f^2, n_f^3, n_f^4)$ is an outward unit normal direction on each face f and $|f|$ is the area for each face f.

Remark 3.4. Using our mesh generation algorithm in Section 2.2, we store the polyhedron in the form of either triangles or squares which leads to the simple formula (3.9). When the faces are general polygons, additional projection operators are needed in order to compute the integral $\int_f \frac{\partial m_j}{\partial n} \phi_i$ (see [10, 11]). $\square$

To compute the stabilization term, we need one more matrix $\boldsymbol{D}_{n_E^v \times 4}$

$$\boldsymbol{D} := \Big(\text{dof}_i(m_j) \Big) = h_E^{-1} \begin{pmatrix} h_E & x_1 - x_E & y_1 - y_E & z_1 - z_E \\ h_E & x_2 - x_E & y_2 - y_E & y_2 - z_E \\ \cdots & \cdots & \cdots & \cdots \\ h_E & x_{n_E^v} - x_E & y_{n_E^v} - y_E & z_{n_E^v} - z_E, \end{pmatrix}$$

where $(x_i, y_i, z_i), i = 1, \cdots, n_E^v$ are vertices in each polyhedron E.

By definition, $\Pi^\nabla v_h = \sum_{\alpha=1}^4 s^\alpha m_\alpha$ and the coefficients (s^α) is determined by the following linear systems

$$(\nabla m_\alpha, \nabla(\Pi^\nabla v_h - v_h))_E = 0 \quad \alpha = 1, \ldots, 4.$$

The matrix representation of $\Pi^\nabla : V_h(E) \to \mathbb{P}_1(E)$ relative to the basis (m_α) is $\boldsymbol{\Pi}^\nabla = \boldsymbol{G}^{-1}\boldsymbol{B}$.

We will also need the matrix representation of Π^∇ in the canonical basis $\{\phi_i\}$. Let $\Pi^\nabla \phi_i = \sum_{j=1}^{n_E^v} \text{dof}_j(\Pi^\nabla \phi_i)\phi_j, i = 1, \cdots, n_E^v$, then the matrix representation $\boldsymbol{\Pi}_*^\nabla$ of the operator $\Pi^\nabla : V_h(E) \to V_h(E)$ in the canonical basis is given by $\boldsymbol{\Pi}_*^\nabla = \boldsymbol{D}\boldsymbol{G}^{-1}\boldsymbol{B} = \boldsymbol{D}\boldsymbol{\Pi}^\nabla$.

Finally the matrix formulation of $\boldsymbol{K}_h^E$ could be written as

$$\boldsymbol{K}_h^E = [\boldsymbol{\Pi}^\nabla]^T \tilde{\boldsymbol{G}} \boldsymbol{\Pi}^\nabla + h_E[\boldsymbol{I} - \boldsymbol{\Pi}_*^\nabla]^T[\boldsymbol{I} - \boldsymbol{\Pi}_*^\nabla],$$

where $\tilde{\boldsymbol{G}}$ is the same with $\boldsymbol{G}$ except that the elements in the first row are all zero.

For the first term of b_i in (3.8), we approximate f by a piecewise constant and approximate

$$(f, \phi_i)_\Omega = \sum_{E \in \Omega_h} (f, \phi_i)_E \approx \sum_{E \in \Omega_h} |E| f(x_E, y_E, z_E)/n_E^v.$$

The second term of b_i could be computed by Gauss quadrature on surface mesh Γ_h.

Remark 3.5. An abstract error estimate of VEM has been given in [10]. With a type of Ceá lemma, the convergence analysis is reduced to the interpolation error estimate $|u - u_I|_1$ and $|u - u_\pi|_{1,E}$, where u_I is the nodal interpolation and u_π is a local approximation of u. Notice that $u_I \in V_h$ is continuous but u_π is most likely discontinuous. To obtain optimal order of the interpolation and approximation error, the authors in [10] further assume the shape-regular condition: there exists a $\gamma > 0$ such that each domain E is star-shaped with respect to a ball of radius $\rho \geq \gamma h_E$, where $h_E = \operatorname{diam}(E)$. This shape regularity assumption will rule out elements generated by our algorithm.

As we mentioned before, for linear finite element space defined on triangles, a refined analysis shows that the optimal first order interpolation error estimate still holds if the maximum angle is uniformly bounded away from π as $h \to 0$ [6]. Such angle condition is generalized to three dimensions, and to high order elements in [76, 3, 37]. Generalization to polyhedron, however, is unknown and under investigation.

3.4 Numerical Experiments

In this section, we present numerical results for the elliptic interface problems in three dimensions. We implement mesh generation and VEM based on the MATLAB® package iFEM [25]. We also solve the algebraic system by an algebraic multigrid (AMG) solver implemented in iFEM [25]. We start with a simple spherical interface and then consider more complex geometric shapes, including two spheres, an orthocircle shape and 12 intersecting spheres. We shall report the following errors:

$$
\begin{aligned}
\|u_I - u_h\|_A &= \left(\|\beta_h^{1/2}\nabla(u_I^- - u_h^-)\|_{\Omega_h^-}^2 + \|\beta_h^{1/2}\nabla(u_I^+ - u_h^+)\|_{\Omega_h^+}^2\right)^{1/2}, \\
\|u_I - u_h\|_\infty &= \max\left\{\|u_I^- - u_h^-\|_{\infty,\Omega_h^-}, \|u_I^+ - u_h^+\|_{\infty,\Omega_h^+}\right\}, \\
\|u_I - u_h\|_{0,h} &= h^{3/2}\left(\sum_{\boldsymbol{x}_i\in\mathcal{N}(\Omega_h^-)} (u_I^-(\boldsymbol{x}_i) - u_h^-(\boldsymbol{x}_i))^2 + \sum_{\boldsymbol{x}_i\in\mathcal{N}(\Omega_h^+)} (u_I^+(\boldsymbol{x}_i) - u_h^+(\boldsymbol{x}_i))^2\right)^{1/2},
\end{aligned}
$$

where u_h is the numerical solution obtained by the linear virtual element methods; u_I^+ and u_I^- are the nodal interpolation of the exact solution u in Ω_h^+ and Ω_h^- respectively. Note that the squared energy norm $\|u_I - u_h\|_A^2$ can be computed by $(u_I^- - u_h^-)^T \boldsymbol{A}_h^-(u_I^- - u_h^-) + (u_I^+ - u_h^+)^T \boldsymbol{A}_h^+(u_I^+ - u_h^+)$ and $\|\cdot\|_{0,h}$ is a good approximation of L^2-norm. The rate is obtained by the least square fitting of the errors in the $\log\log$ scale.

Example 3.1 (One sphere)**.** The domain Ω is $(-1,1)^3$ and the interface is defined by $\phi(x,y,z) = x^2+y^2+z^2-r^2$ with radius $r = 0.75$. The coefficient β is piecewise constant. The analytic solution is given by $u^+ = 10(x+y+z)$ and $u^- = 5\exp(x^2+y^2+z^2)+20$. In this case, the solution is discontinuous and the flux jump across the interface is also non-homogenous.

Fig. 3.1 shows the surface mesh extracted from the volume mesh generated by our algorithm for the spherical interface. The maximal interior angle of triangular faces on the surface mesh is bounded by $112.8104°$. Tables 3.1 and 3.2 show the error for $\beta^- = 1, \beta^+ = 10$ and $\beta^- = 1, \beta^+ = 100$, respectively. As it can be seen that near second order accuracy is attained in both $\|\cdot\|_{0,h}$ and

$\|\cdot\|_\infty$ norms. The convergence rate in the energy norm is near 1.5, which is consistent with the superconvergence result obtained in [24]. This superconvergence occurs due to the nice properties of our semi-unstructured mesh. It seems that the convergence rate is robust to the variation of β.

From Table 3.3, we can conclude that the runtimes of the mesh generation part can be ignored compared with the assembling and solving parts. In Table 3.4, we present the variation of iteration steps of the algebraic multigrid method with respect to the number of degrees of freedom (#dof) and to the variation of jump coefficients (fix $\beta^- = 1$ and change β^+). It indicates that the algebraic multigrid method is a robust and efficient solver: robust to the number of degrees of freedom and to the variation of jump coefficients.

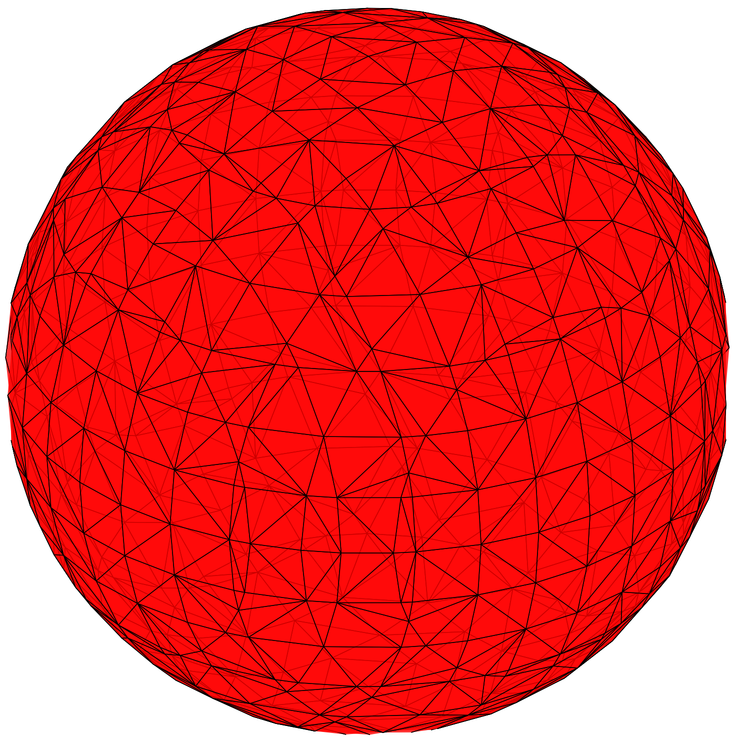


Figure 3.1: An interface mesh with maximal angle $112.8104°$.

Table 3.1: Errors for Example 3.1: $\beta^- = 1$ and $\beta^+ = 10$.

#dof	h	$\|\|u_I - u_h\|\|_A$	$\|\|u_I - u_h\|\|_\infty$	$\|\|u_I - u_h\|\|_0$
10,323	0.1	4.45798e-01	5.24227e-02	3.62347e-02
72,713	0.05	1.64215e-01	1.81762e-02	8.97415e-03
547,881	0.025	6.62120e-02	5.09725e-03	2.62269e-03
4,240,529	0.0125	2.43899e-02	1.37664e-03	7.05429e-04
Rate		1.4	1.8	1.9

Table 3.2: Errors for Example 3.1: $\beta^- = 1$ and $\beta^+ = 100$.

#dof	h	$\|\|u_I - u_h\|\|_A$	$\|\|u_I - u_h\|\|_\infty$	$\|\|u_I - u_h\|\|_0$
10,323	0.1	6.65319e-01	5.14276e-02	3.40871e-02
72,713	0.05	2.41564e-01	1.84007e-02	8.42552e-03
547,881	0.025	8.98117e-02	5.21504e-03	2.50271e-03
4,240,529	0.0125	3.21041e-02	1.42298e-03	6.80724e-04
Rate		1.5	1.7	1.9

Table 3.3: CPU time (in seconds) for Example 3.1: $\beta^- = 1$ and $\beta^+ = 10$.

#dof	Assemble	Solve	Mesh
10,323	0.228973	0.47	0.21165
72,713	0.970122	2.59	0.4854
547,881	6.99419	13.55	1.8981
4,240,529	62.1644	121.59	8.0172

Table 3.4: Example 3.1: Iteration steps of AMG with fixed $\beta^- = 1$ and various β^+.

β^+ / #dof	10^{-3}	10^{-2}	10^{-1}	1	10	10^2	10^3	10^4
7,921	10	10	10	9	9	9	9	9
63,111	11	11	11	11	11	10	10	10
509,479	12	12	12	12	14	13	13	13
4,086,927	13	12	13	15	17	16	16	16

EXAMPLE 3.2 (Two spheres)**.** The domain Ω is $(-1,1)^3$ and the interface is defined by

$$\phi(x,y,z) = \min\{\phi_1, \phi_2\}$$

where

$$\phi_1 = (x+0.5r)^2 + (y+0.5r)^2 + (z+0.75r)^2 - r^2,$$
$$\phi_2 = (x-0.25r)^2 + (y-0.75r)^2 + (z-r)^2 - r^2,$$

with radius $r = 0.4$. The coefficient β is piecewise constant. The analytic solution is given by $u^+ = 10(x^2+y^2+z^2)$ and $u^- = 5\cos(x^2+y^2+z^2)$.

Fig. 3.2 shows that two spheres are embedded in the unit cube. When $h = 0.1$, i.e., the background Cartesian grid is not fine enough, there exists an interface element which is divided into three parts by these two spheres. The maximal interior angle of triangular faces on the surface mesh is bounded by $130.4665°$. Tables 3.5 and 3.6 show the numerical results for $\beta^- = 1, \beta^+ = 1$ and $\beta^- = 1, \beta^+ = 100$, respectively. Table 3.7 shows how the computational time grows with respect to the number of degrees of freedom. Table 3.8 shows the number of iterations taken by the algebraic multigrid method for various values of input parameters.

All results are consistent with our conclusion. It indicates that our algorithm works in a case when the interface is unconnected.

Table 3.5: Errors for Example 3.2: $\beta^- = 1$ and $\beta^+ = 1$.

#dof	h	$\|\|u_I - u_h\|\|_A$	$\|\|u_I - u_h\|\|_\infty$	$\|\|u_I - u_h\|\|_0$
9,789	0.1	3.76723e-01	1.08148e-01	6.52076e-02
71,225	0.05	1.32276e-01	2.82699e-02	1.74225e-02
540,945	0.025	4.52335e-02	7.30380e-03	4.28321e-03
4,211,729	0.0125	1.60165e-02	1.92081e-03	1.11271e-03
Rate		1.5	1.9	2

Table 3.6: Errors for Example 3.2: $\beta^- = 1$ and $\beta^+ = 100$.

#dof	h	$\|\|u_I - u_h\|\|_A$	$\|\|u_I - u_h\|\|_\infty$	$\|\|u_I - u_h\|\|_0$
9,789	0.1	3.67145e+00	1.15596e-01	5.54995e-02
71,225	0.05	1.35081e+00	3.08549e-02	1.51136e-02
540,945	0.025	4.78640e-01	8.22331e-03	3.74778e-03
4,211,729	0.0125	1.72934e-01	2.25395e-03	9.77991e-04
Rate		1.5	1.9	2

Table 3.7: CPU time (in seconds) for Example 3.2: $\beta^- = 1$ and $\beta^+ = 1$.

#dof	Assemble	Solve	Mesh
9,789	0.160716	1.01	0.259938
71,225	0.706959	5.84	0.36018
540,945	6.32812	27.26	1.78691
4,211,729	52.7369	133.86	10.067

Table 3.8: Example 3.2: Iteration steps of AMG with fixed $\beta^- = 1$ and various β^+.

β^+ / #dof	10^{-3}	10^{-2}	10^{-1}	1	10	10^2	10^3	10^4
7,387	10	11	10	9	9	9	9	9
61,623	11	11	11	10	10	10	10	10
502,543	12	12	12	12	12	11	12	11
4,058,127	13	13	13	12	13	13	13	13

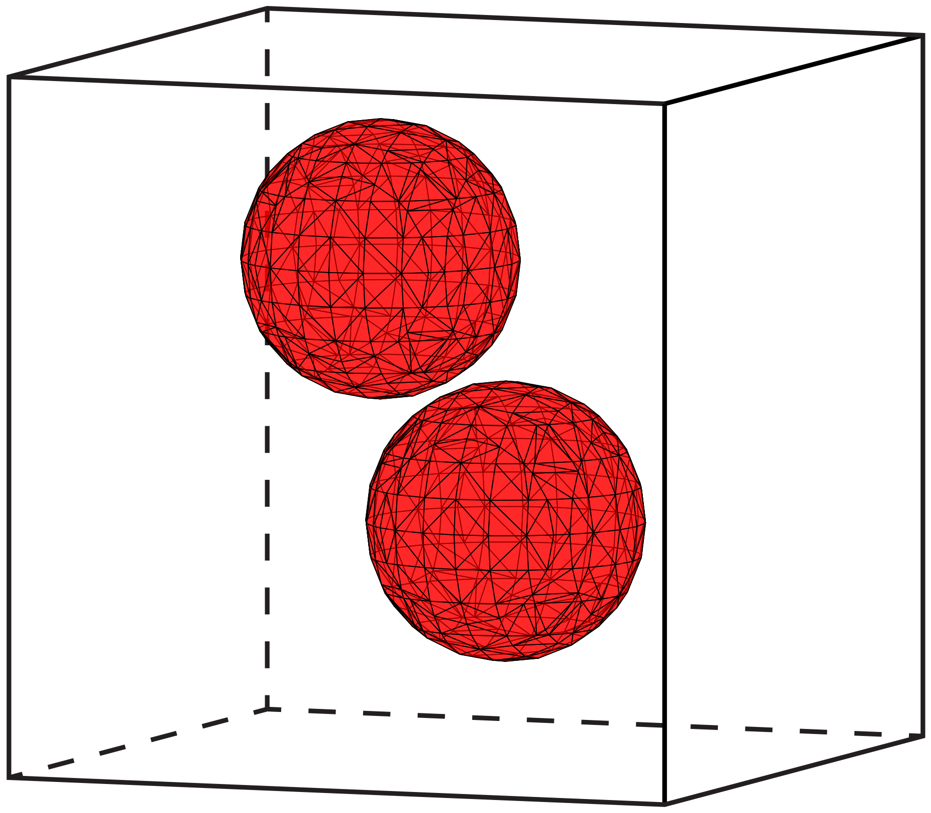


Figure 3.2: Two balls are embedded in the unit cube. The maximal angle is $130.4665°$.

EXAMPLE 3.3 (An orthocircle)**.** The domain Ω is $(-1.2, 1.2)^3$ and the interface is defined by

$$\phi(x,y,z) = \left[(x^2+y^2-1)^2+z^2\right]\left[(x^2+z^2-1)^2+y^2\right]\left[(y^2+z^2-1)^2+x^2\right] - 0.075^2 \left[1+3(x^2+y^2+z^2)\right].$$

The coefficient β is piecewise constant. The analytic solution is given by $u^+ = 1 - x^2 - y^2 - z^2$ and $u^- = \sin(\pi x)\sin(\pi y)\sin(\pi z)$.

Fig. 3.3 shows the interface-fitted mesh extracted as the boundary of Ω_h^-. The maximal angle of triangular faces of the interface mesh is bounded by $132.4673°$. Tables 3.9 and 3.10 show the numerical results for $\beta^- = 1, \beta^+ = 1$ and $\beta^- = 1, \beta^+ = 100$, respectively. Table 3.11 shows the computational time accordingly. The mesh part is still quick even the interface with complex geometry. Table 3.12 is the number of iterations of the algebraic multigrid (AMG) solver.

These test results indicate that the proposed interface problem solver works well even for complex surfaces. Note that the maximal angles of the surface mesh in our examples are uniformly bounded by $144°$, then there is no need to be smoothed as a post-processing step.

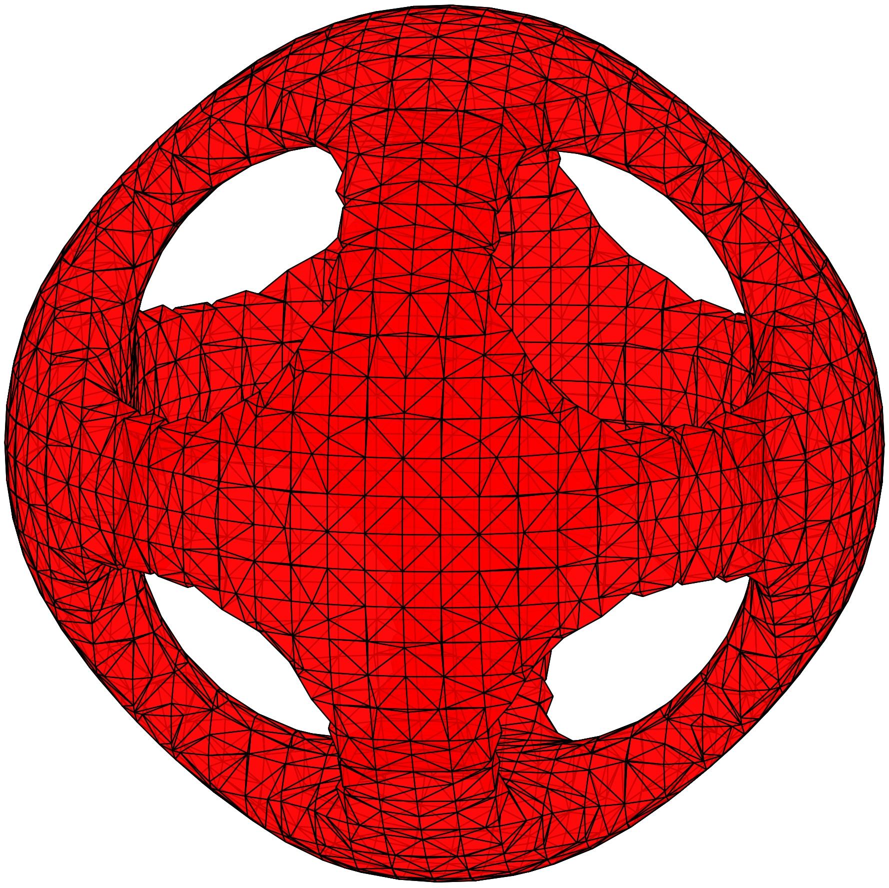


Figure 3.3: The interface is an orthocircle with maximal angle $132.4673°$.

EXAMPLE 3.4 (12 intersecting spheres)**.** In this example, we consider a more complicated interface formed by 12 intersecting spheres, which has many one-dimension sharp features. The domain Ω is $(-3,3)^3$ and the interface is defined by

$$\phi(x,y,z) = \min\{\phi_1, \phi_2, \cdots, \phi_{12}\}$$

Table 3.9: Errors for Example 3.3: $\beta^- = 1$ and $\beta^+ = 1$.

#dof	h	$\|\|u_I - u_h\|\|_A$	$\|\|u_I - u_h\|\|_\infty$	$\|\|u_I - u_h\|\|_0$
11,145	0.12	2.93834e-01	5.50055e-02	5.14120e-02
76,469	0.06	7.95795e-02	1.05921e-02	4.18525e-03
561,957	0.03	2.35627e-02	2.22961e-03	9.17810e-04
4,295,165	0.015	7.80143e-03	6.05301e-04	2.15878e-04
Rate		1.7	2.1	2.1

Table 3.10: Errors for Example 3.3: $\beta^- = 1$ and $\beta^+ = 100$.

#dof	h	$\|\|u_I - u_h\|\|_A$	$\|\|u_I - u_h\|\|_\infty$	$\|\|u_I - u_h\|\|_0$
11,145	0.12	3.44042e+00	9.63266e-02	1.24556e-01
76,469	0.06	7.72426e-01	1.21513e-02	1.56542e-02
561,957	0.03	2.06488e-01	3.04186e-03	3.34684e-03
4,295,165	0.015	6.30787e-02	7.45803e-04	8.10506e-04
Rate		1.8	2	2.1

Table 3.11: CPU time (in seconds) for Example 3.3: $\beta^- = 1$ and $\beta^+ = 1$.

#dof	Assemble	Solve	Mesh
11,145	0.493225	0.75	0.383038
76,469	1.41325	1.72	0.857616
561,957	8.1635	12.05	3.48605
4,295,165	62.6127	97.92	13.8495

Table 3.12: Example 3.3: Iteration steps of AMG with fixed $\beta^- = 1$ and various β^+.

β^+ / #dof	10^{-3}	10^{-2}	10^{-1}	1	10	10^2	10^3	10^4
8,743	15	14	15	13	11	10	10	10
66,867	13	12	11	11	11	11	11	11
523,555	13	12	12	12	12	12	12	12
4,141,563	13	13	13	13	13	13	13	13

where $\phi_i, i = 1, \dots, 12$ are equal radius balls with centers

$$(1.0, 0, 0), (-1.0, 0, 0), (0.5, 0.866025403784439, 0), (-0.5, 0.866025403784439, 0),$$
$$(0.5, -0.866025403784439, 0), (-0.5, -0.866025403784439, 0), (2.0, 0, 0), (-2.0, 0, 0),$$
$$(1.0, 1.73205080756888, 0), (-1.0, 1.73205080756888, 0), (-1.0, -1.73205080756888, 0),$$
$$(1.0, -1.73205080756888, 0)$$

and radius $r = 0.7$. The coefficient β is a piecewise constant. The analytic solution is given by $u^+ = 10(x^2 + y^2 + z^2)$ and $u^- = 5\cos(x^2 + y^2 + z^2)$.

Fig. 3.4 shows the interface-fitted mesh extracted as the boundary of Ω_h^-, which did not capture the sharp one-dimension features of the interface, and this will lead to a loss of the solution accuracy. From part of the interface-fitted mesh, there are geometric singularities with sharp edges at the intersection of balls. See Fig. 3.5 for an illustration. In order to capture the features of complicated interface and improve the solution accuracy, one need to use adaptive mesh near the geometric features, and this will be our future work. But the maximal angle condition is still satisfied as the maximal angle of triangular faces on the surface mesh is bounded by $131.3925°$. Tables 3.13 and 3.14 show the numerical results for $\beta^- = 1, \beta^+ = 1$ and $\beta^- = 1, \beta^+ = 10$, respectively. As it can be seen that the convergence rate in the energy norm is still near 1.5. But the convergence rates in the $\|\cdot\|_{0,h}$ and $\|\cdot\|_\infty$ norms are not second order due to geometric singularities. Table 3.15 shows how the computational time grows with respect to the number of degrees of freedom. Table 3.16 shows the number of iterations taken by the algebraic multigrid method for various values of input parameters.

Table 3.13: Errors for Example 3.4: $\beta^- = 1$ and $\beta^+ = 1$.

#dof	h	$\|\|u_I - u_h\|\|_A$	$\|\|u_I - u_h\|\|_\infty$	$\|\|u_I - u_h\|\|_0$
2,527	0.48	2.27225e+01	3.60464e+00	6.97070e+00
18,960	0.24	1.18700e+01	2.24845e+00	4.20002e+00
138,051	0.12	3.99601e+00	8.79537e-01	9.22094e-01
1,051,665	0.06	1.57695e+00	2.91154e-01	3.92484e-01
Rate		1.4	1.2	1.6

Table 3.14: Errors for Example 3.4: $\beta^- = 1$ and $\beta^+ = 10$.

#dof	h	$\|\|u_I - u_h\|\|_A$	$\|\|u_I - u_h\|\|_\infty$	$\|\|u_I - u_h\|\|_0$
2,527	0.48	8.71247e+01	5.29705e+00	8.63843e+00
18,960	0.24	3.90973e+01	3.07134e+00	4.54532e+00
138,051	0.12	1.38105e+01	1.37945e+00	1.15474e+00
1,051,665	0.06	5.49027e+00	4.88589e-01	3.78441e-01
Rate		1.3	1.1	1.6

Table 3.15: CPU time (in seconds) for Example 3.4: $\beta^- = 1$ and $\beta^+ = 1$.

#dof	Assemble	Solve	Mesh
2,527	0.213214	0.29	0.3674
18,960	0.431979	1.81	0.61863
138,051	2.22761	4.88	2.4452
1,051,665	29.9028	42.76	6.8316

Table 3.16: Example 3.4: Iteration steps of AMG with fixed $\beta^- = 1$ and various β^+.

β^+ / #dof	10^{-2}	10^{-1}	1	10	10^2	10^3	10^4
1,611	11	10	9	8	9	8	8
15,208	12	11	10	10	10	10	10
123,049	12	12	11	11	11	11	11
991,663	14	12	12	12	12	12	12

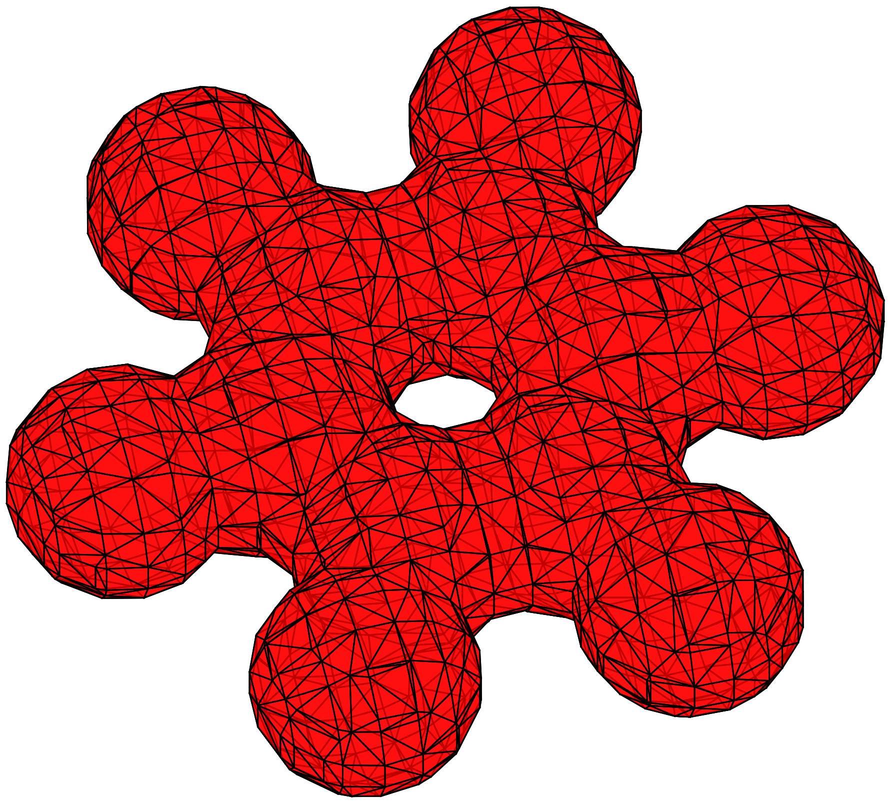


Figure 3.4: Twelve balls are embedded in the cube. The maximal angle is $131.3925°$.

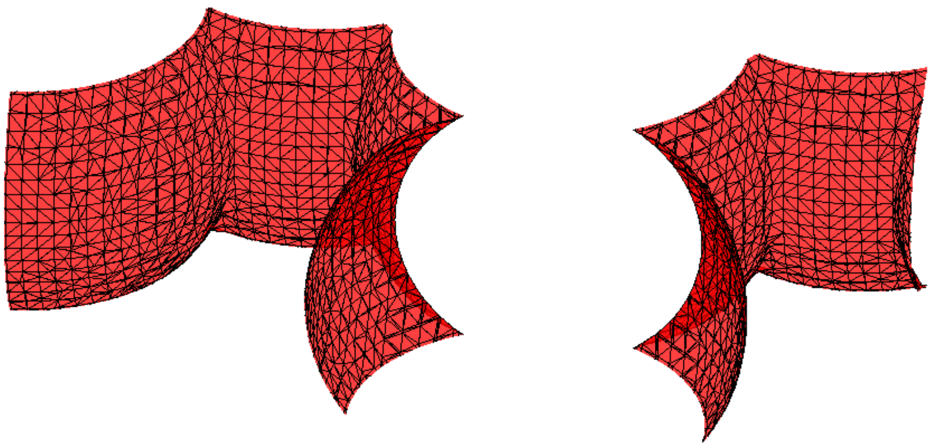


Figure 3.5: Part of the interface-fitted mesh with sharp edges.

Chapter 4

Moving Interface Problems

There are two major sections in this chapter. The first section is to discuss the 2D model of the tissue growth problems. The second section is about the 2D model of the tumor growth problems. In this chapter we will discuss the applications of solving the Poisson equation with a moving boundary domain. Free interface problems have been applied heavily in different fields such as biology, physics, and engineering recently. However, due to the discontinuities of physical quantities across the interface designing accurate numerical methods for such problems is still challenging. The fundamental problem in exploring the numerical scheme is how to treat the discontinuities. Many schemes have tried to use the different strategies to handle the discontinuities across the interface. For example, one of the most popular strategies is to convert the discontinuities to singular terms by adding the regularization term. The method is very straightforward and robust with various spatial discretization methods such as finite difference, finite element, and the finite volume method, and so on. Level-set method [102], immersed boundary method [106, 107] and front tracking methods [112] use this approach. Another popular strategy is to use local differencing schemes or basis functions near the interface to approximate the discontinuities. The matched interface and boundary method (MIB) employed this strategy [145, 146]. The last popular strategy applied interface-fitted mesh with interfaces passing through mesh node points [113]. Here, we use the interface-fitted mesh strategy to solve the moving interfaced problems since we could generate good quality mesh efficiently. In every time step of the interface evolution, we use the zero level set of the level set function to identify the interface position and then generate the interface-fitted mesh. The velocity of the flow is solved using the linear finite element method and then used it to solve the transport equation to update the level set function. We iterate this process to evolve the interface. In order to show our schemes in details, we applied our schemes to solve the interface problems in biology.

4.1 Tissue Growth Problem

This section is to introduce the tissue growth problem with static interface at first. The linear and quadratic element methods are showed for this case. The moving interface problem is discussed afterwards.

4.1.1 Tissue Growth Model

The following model describes the process of tissue growth [103]. The boundary is either the interface between neighboring tissues or an open external environment and the pressure on the boundary is proportional to the curvature of the boundary. The model has periodic boundary condition along the x-direction throughout the tissue. The growth of tissue is quantified by the dynamic boundary of the system. First, we discuss derivation of equations, which is on the two dimensional rectangle domain $\Omega = (0, 2) \times (0, h)$.

$$-K\Delta P = \Psi, x \in \Omega \tag{4.1}$$

with boundary conditions:

$$v(x, y = 0, t) = \frac{\partial P}{\partial y}(x, y = 0, t) = 0, \tag{4.2}$$

$$P(x, h, t) = \xi\kappa = \xi\frac{\partial^2 h/\partial x^2}{[1 + (\partial h/\partial x)^2]^{3/2}}, \tag{4.3}$$

and the periodic boundary condition:

$$P(x = 0, y, t) = P(x = 2, y, t). \tag{4.4}$$

The growth of the dynamic boundary of the system is determined by h, which is governed by the kinematic condition,

$$\frac{\partial h}{\partial t} + u(x, h, t)\frac{\partial h}{\partial x} = v(x, h, t).$$

Here $u = -KP_x$ and $v = -KP_y$ [103], P is the pressure, the gradient obtained from (4.1) (P_x, P_y) gives the velocity of the tissue, Ψ represents the net rate of production, ξ is related to the magnitude of the surface tension along the boundary and K is the permeability. See Fig. 4.1 for illustration.

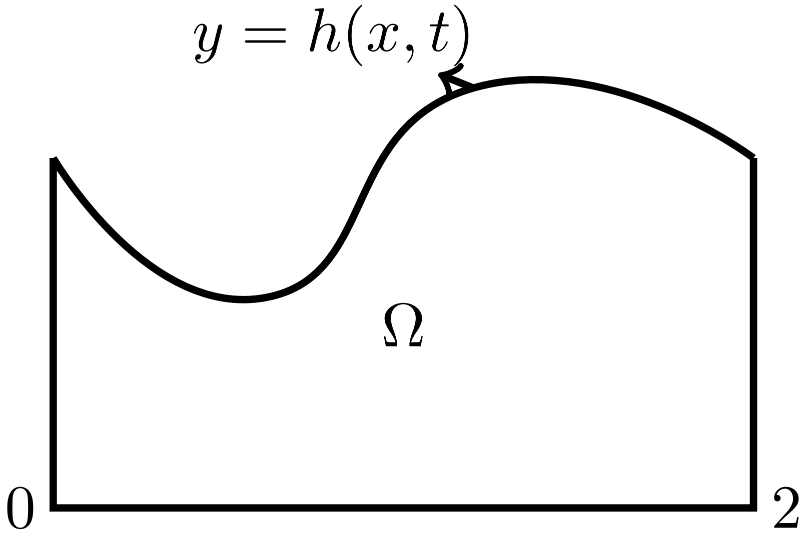


Figure 4.1: The domain of tissue growth model.

4.1.2 Kinematic Boundary Condition

In order to link the velocity of a boundary to the particle adjacent to the boundary we need to impose a condition. It is the motivation of the kinematic boundary condition, which is modeled by the transport equation. Firstly we consider the general 2D equations for a free-surface flow with the compressible fluid and the flow driven by the potential.

$\mathbf{u} = \nabla P$ and $-\Delta P = f$ are used to describe the fluid, where $\mathbf{u} = (u, w)$ and $P = P(x, y, t)$. $y = h(x, t)$ is the surface of the fluid in motion, f is the force term and $y = 0$ is the lower boundary where the vertical velocity vanishes.

$$0 = \mathbf{n} \cdot \nabla p = \frac{\partial p}{\partial y} = v.$$

In other words, we get

$$\frac{\partial p}{\partial y} = 0 \quad \text{on} \quad y = 0.$$

The velocity of the fluid normal to the boundary is required to be equal to the velocity of the boundary normal to itself. Otherwise, the fluid would either be flowing through the boundary or separating from it. Neither of them is acceptable. Therefore, the kinematic boundary condition on a moving surface is $\frac{DS}{Dt} = 0$ and the free surface is the zero set of $S(\mathbf{x}, t)$. We define a function based on the height function $h(x, t)$.

$$S(x, y, t) = y - h(x, t),$$

It is zero at the free surface. Thus

$$0 = (\frac{\partial}{\partial t} + u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y})(y - h(x, t)) = -\frac{\partial h}{\partial t} - \frac{\partial p}{\partial x}\frac{\partial h}{\partial x} + \frac{\partial p}{\partial y} \quad \text{on} \quad y = h.$$

Therefore, the kinematic boundary condition holds

$$\frac{\partial h}{\partial t} + u\frac{\partial h}{\partial x} = v \quad \text{on} \quad y = h. \tag{4.5}$$

We chose the discontinuous Galerkin finite element (DG-FEM) scheme to solve the above transport equation 4.5. By subdividing the interval $\Omega = [L, R]$ into a union of non-overlapping elements $D^k = [x_l^k, x_r^k]$

$$\Omega \approx \Omega_h = \cup_{k=1}^{K} D^k,$$

as illustrated in Fig. 4.2 .

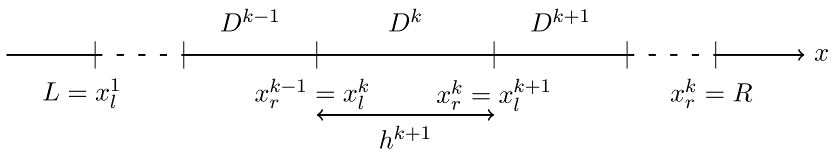


Figure 4.2: Sketch of the geometry for simple one-dimensional example.

Consider the one-dimensional scalar conservation law for the solution $u(x,t)$

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = g. \tag{4.6}$$

subject to an appropriate set of initial conditions $u(x, 0) = u_0(x)$ and boundary conditions on the boundary, $\partial\Omega$, where $f(u)$ is the flux and $g(x,t)$ is some prescribed forcing function. To solve Eq. 4.6, we define a space of test functions, V_h, to which the residual is orthogonal to in this space in the weak sense as

$$\int_{\Omega}(\frac{\partial u_h}{\partial t} + \frac{\partial f_h}{\partial x} - g_h)\phi_h(x)dx = 0, \quad \forall \phi_h \in V_h.$$

Next, the local residual is formed on the $k = 1, \cdots, NE$ elements

$$x \in D^k : R_h^k(x,t) = \partial_t u_h^k + \partial_x f_h^k - g_h^k$$

and require this to vanish locally in a Galerkin sense

$$\int_{D^k} R_h^k(x,t) l_i^k(x) dx = 0, i = 1, \ldots, N_p, k = 1, \ldots, NE$$

It forms the basis of a nodal DG-FEM scheme. Gauss's theorem is applied to obtain the local statement.

$$\int_{D^k} \frac{\partial u_h^k}{\partial t} l_j^k - f_h^k \frac{dl_j^k}{dx} - g l_j^k dx = -[f_h^k l_j^k]_{x^k}^{x^{k+1}}.$$

The right-hand side is used to connect the elements. However, the uniqueness of the solutions at interfaces between adjacent elements cannot meet. To overcome it a numerical flux f^* is employed to approximate the physical flux. We applied Gauss's theorem and get

$$\int_{D^k} \frac{\partial u_h^k}{\partial t} l_j^k - f_h^k \frac{dl_j^k}{dx} - g l_j^k dx = -[f^* l_j^k]_{x^k}^{x^{k+1}}.$$

Then we applied Gauss's theorem once again,

$$\int_{D^k} R_h(x,t) l_j^k dx = [(f_h^k - f^*) l_j^k]_{x^k}^{x^{k+1}}.$$

The two schemes are the weak and strong forms for the scalar conservation law. The key of the scheme is how to choose the numerical flux, f^*. The two local elementwise schemes are written using the terminology of the finite element scheme.

$$\boldsymbol{M^k}\frac{d\boldsymbol{u}_h^k}{dt} - (\boldsymbol{S^k})^T\boldsymbol{f}_h^k - \boldsymbol{M^k}\boldsymbol{g}_h^k = -f^*(x^{k+1})l^k(x^{k+1}) + f^*(x^k)l^k(x^k),$$

and

$$\boldsymbol{M^k}\frac{d\boldsymbol{u}_h^k}{dt} + \boldsymbol{S^k}\boldsymbol{f}_h^k - \boldsymbol{M^k}\boldsymbol{g}_h^k = (f_h^k(x^{k+1}) - f^*(x^{k+1}))l^k(x^{k+1}) - (f_h^k(x^k) - f^*(x^k))l^k(x^k).$$

Here local unknowns,$\boldsymbol{u}_h^k$,of fluxes,$\boldsymbol{f}_h^k$, and the forcing,$\boldsymbol{g}_h^k$ are given on the nodes in each element. $\boldsymbol{l}^k(x) = [l_1^k(x), \ldots, l_{N^p}^k(x)]^T$, and $\boldsymbol{M}_{ij}^k = \int_{D^k} l_i^k(x)l_j^k(x)dx$ and $\boldsymbol{S}_{ij}^k = \int_{D^k} l_i^k(x)\frac{dl_j^k(x)}{dx}dx$ are the local mass matrix and stiffness matrix, respectively [58].

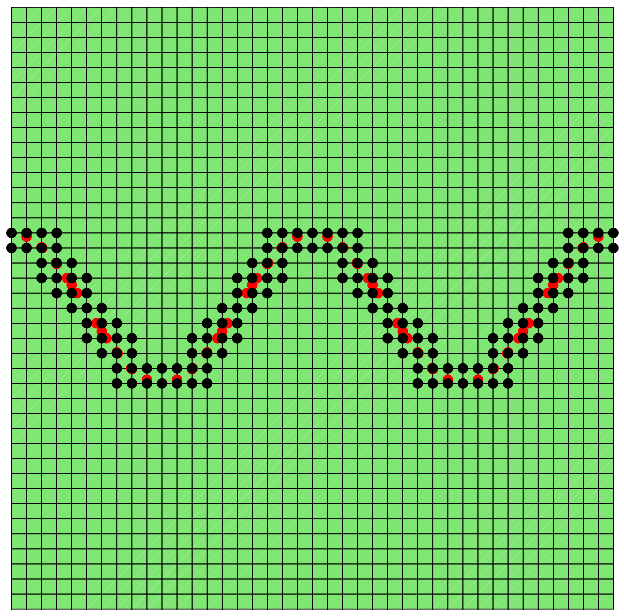
We construct the matrices of $\boldsymbol{M}_{ij}^k$ and $\boldsymbol{S}_{ij}^k$ as the similar way of assembling the mass and the stiffness matrices in the finite element methods.

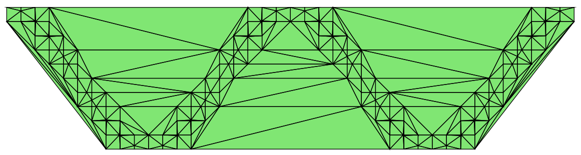
4.1.3 Static Problems

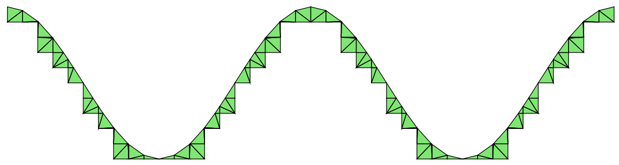
We consider the static problem to test our scheme first and then solve the problem using linear and quadratic finite element methods. K and ξ are set as 1. The top boundary of the tissue is given by $h(x) = 1 + B\cos(2\pi x)$ with different B for numerical tests. The exact solution of the pressure is

$$P(x,h) = \frac{-4\pi^2 B\cos(2\pi x)\cos(2\pi y)/(1 + B\cos(2\pi x))}{[1 + 4\pi^2B^2\sin^2(2\pi x)]^{3/2}}.$$

We need a larger box containing the domain Ω and generate the mesh for the rectangle box. In the first step, we could get the interface points based on given Δx. In the second step, we generate Delaunay triangulations on interface points. In the last step, we remove triangles not in the interface elements and merge all uncut elements below h to get a semi-unstructured mesh. Finally, we could get the approximated mesh of Ω. Figure 4.3 shows the mesh generation procedure. In order to reduce the complexity, we could divide all squares into triangles (see Figure 4.4) and use finite element methods to solve the Poisson equations.


(a) Step 1: Find all the interface points on the curve h.


(b) Step 2: a Delaunay triangulation on interface points.


(c) Step 3: Remove triangles not in the interface elements.

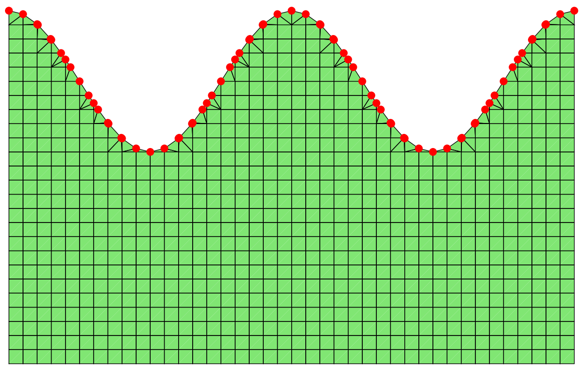

(d) Step 3: Merge all uncut elements to get a semi-unstructured mesh.

Figure 4.3: Three steps to generate a semi-unstructured mesh.

Based on the assumption in the X direction of the domain, there is a straightforward way to impose periodic boundary conditions in the matrix form. The details are shown in [47]. It is, however, not easy to implement the method for higher order schemes. In this thesis, we use an alternative way to impose the periodic boundary conditions, which could be used for higher order algorithm easily. At first, we get the original mesh as above, which is only used for all geometric information such

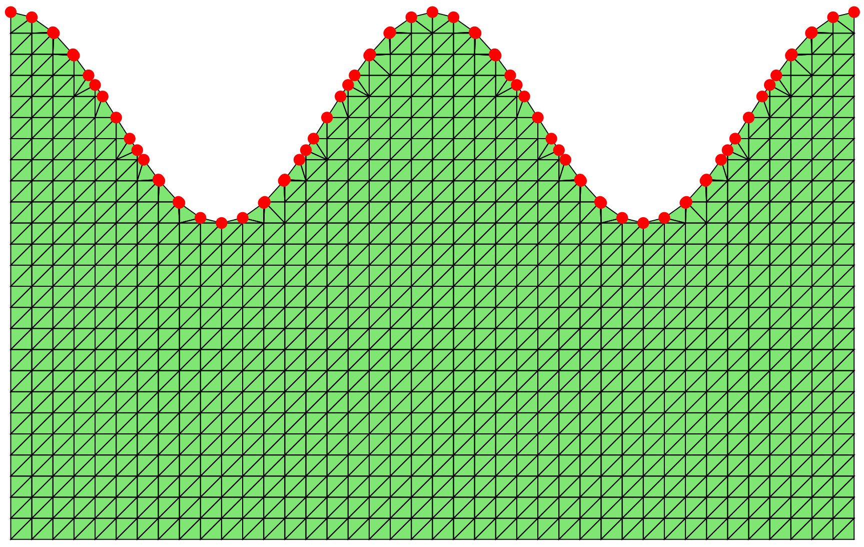
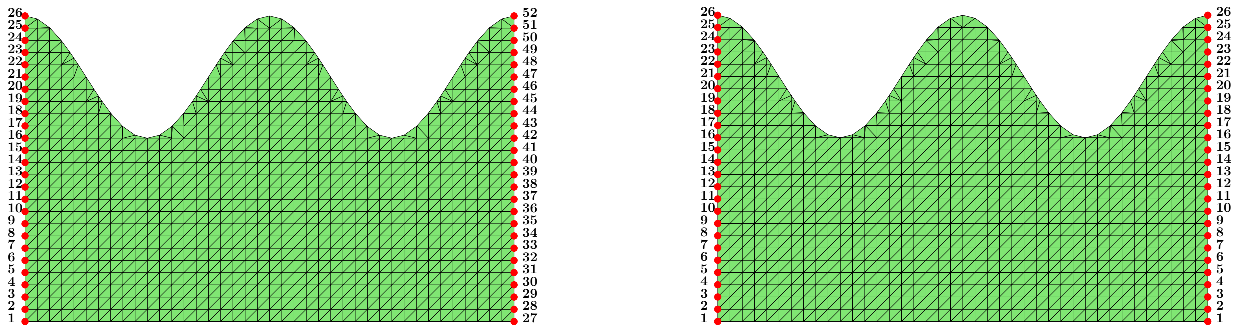


Figure 4.4: All elements are triangles

as length, area. See the right graph in Figure 4.5. Then we introduce another artificial mesh which requires that the index of nodes on right hand side of the points is the same as that of the left hand side of the points. See the left graph in Figure 4.5 for illustration. It is used for assembling the finite element matrix except geometric information. For linear finite element methods, we use a map between the original mesh and artificial mesh called `nodeMap` to impose periodic conditions. For quadratic finite element methods, another map named `dofMap` should be introduced to get the relationship between middle points. It could be extended to any higher order elements.



(a) The original mesh based on initial boundary condition of h.

(b) Construct the artificial mesh which imposes periodic conditions

Figure 4.5: The mesh with periodic boundary condition

In this subsection, we test the convergence rates to demonstrate the effectiveness of our methods.

The energy ($||\cdot||_A$) and maximum norms ($||\cdot||_\infty$) are considered. Here is the definition:

$$||P_I - P_{\Delta x}||_A = \sqrt{(P_I - P_{\Delta x})^T \boldsymbol{A}(P_I - P_{\Delta x})}, \tag{4.7}$$

$$||P_I - P_{\Delta x}||_\infty = \max|P_I - P_{\Delta x}|, \tag{4.8}$$

where P_I is the nodal interpolation of the exact solution of P, $P_{\Delta x}$ is the numerical solution obtained by finite element methods, $\boldsymbol{A}$ is the corresponding stiffness matrix and Δx is the mesh size of the initial Cartesian mesh.

EXAMPLE 4.1 ($B = 0.1$)**.** Table 4.1 and 4.2 show the numerical convergence results for linear and quadratic finite element methods respectively when $B = 0.1$.

Table 4.1: Errors for Example 1: linear finite element methods when $B = 0.1$.

#Dof	Δx	$\|\|P_I - P_{\Delta x}\|\|_A$	$\|\|P_I - P_{\Delta x}\|\|_\infty$
455	5.00e-02	2.09603e-01	7.44661e-02
1715	2.50e-02	5.58425e-02	2.00167e-02
6635	1.25e-02	1.50097e-02	4.96067e-03
26075	6.25e-03	4.41325e-03	1.21513e-03
Rate		1.9	2

Table 4.2: Errors for Example 1: quadratic finite element methods when $B = 0.1$.

#Dof	Δx	$\|\|P_I - P_{\Delta x}\|\|_A$	$\|\|P_I - P_{\Delta x}\|\|_\infty$
887	5.00e-02	5.82090e-02	3.14154e-03
3385	2.50e-02	8.69113e-03	4.42525e-04
13181	1.25e-02	1.34846e-03	5.94591e-05
51973	6.25e-03	2.21733e-04	7.68399e-06
206357	3.13e-03	3.71963e-05	9.75976e-07
Rate		2.7	2.9

EXAMPLE 4.2 ($B = 0.15$)**.** Table 4.3 and 4.4 show the numerical results for linear and quadratic finite element methods respectively when $B = 0.15$.

Table 4.3: Errors for Example 2: linear finite element methods when $B = 0.15$.

#Dof	Δx	$\|\|P_I - P_{\Delta x}\|\|_A$	$\|\|P_I - P_{\Delta x}\|\|_\infty$
460	5.00e-02	5.86786e-01	1.43121e-01
1725	2.50e-02	1.45334e-01	4.35102e-02
6655	1.25e-02	3.80448e-02	1.11352e-02
26115	6.25e-03	1.00820e-02	2.72108e-03
Rate		2	2

Table 4.4: Errors for Example 2: quadratic finite element methods when $B = 0.15$.

#Dof	Δx	$\|\|P_I - P_{\Delta x}\|\|_A$	$\|\|P_I - P_{\Delta x}\|\|_\infty$
896	5.00e-02	1.68363e-01	1.02863e-02
3403	2.50e-02	2.53524e-02	1.31378e-03
13217	1.25e-02	3.88118e-03	1.85960e-04
52045	6.25e-03	6.20807e-04	2.48831e-05
206501	3.13e-03	1.04426e-04	3.19088e-06
Rate		2.7	2.9

EXAMPLE 4.3 ($B = 0.25$)**.** Table 4.5 and 4.6 show the numerical results for linear and quadratic finite element methods respectively when $B = 0.25$.

Table 4.5: Errors for Example 3: linear finite element methods when $B = 0.25$.

#Dof	Δx	$\|\|P_I - P_{\Delta x}\|\|_A$	$\|\|P_I - P_{\Delta x}\|\|_\infty$
470	5.00e-02	4.02367e+00	6.86283e-01
1745	2.50e-02	6.66825e-01	1.31735e-01
6695	1.25e-02	1.66809e-01	3.81307e-02
26195	6.25e-03	4.26586e-02	9.52049e-03
Rate		2.2	1.9

EXAMPLE 4.4 ($B = 0.5$)**.** Table 4.7 and 4.8 shows the numerical results for linear and quadratic finite element methods respectively when $B = 0.5$.

We could get the conclusion that near second order of accuracy of our method is in $||\cdot||_A$ and $||\cdot||_\infty$ norm for linear finite element methods. We also consider high order finite element methods such

Table 4.6: Errors for Example 3: quadratic finite element methods when $B = 0.25$.

#Dof	Δx	$\|\|P_I - P_{\Delta x}\|\|_A$	$\|\|P_I - P_{\Delta x}\|\|_\infty$
914	5.00e-02	8.47067e-01	6.98431e-02
3439	2.50e-02	1.30214e-01	6.88447e-03
13289	1.25e-02	1.94364e-02	9.88083e-04
52189	6.25e-03	3.01006e-03	1.34961e-04
206789	3.13e-03	4.91737e-04	1.77970e-05
Rate		2.7	2.9

Table 4.7: Errors for Example 4: linear finite element methods when $B = 0.5$.

#Dof	Δx	$\|\|P_I - P_{\Delta x}\|\|_A$	$\|\|P_I - P_{\Delta x}\|\|_\infty$
495	5.00e-02	1.08631e+02	4.41181e+01
1795	2.50e-02	9.94629e+00	1.39314e+00
6795	1.25e-02	1.68044e+00	2.27722e-01
26395	6.25e-03	4.15898e-01	6.59658e-02
Rate		2.7	2.2

as quadratic finite element methods. The convergence rates for $||\cdot||_A$ and $||\cdot||_\infty$ norms are nearly three. It seems that the convergence rate is robust to the variability of the coefficient B. But the iterative pseudo spectral approach fails to converge for some B [103]. What's more, our method could be extended to any higher order algorithm easily. The convergence rate in the energy norm is higher than the theoretical results due to the nice structure of our grids. And this is known as superconvergence.

4.1.4 Moving Problems

In this subsection, we test the accuracy of our approach for the moving interface problems with known or unknown solutions. We use linear finite element methods and second order of DG-FEM to solve the coupled system of both P and h. In addition, a 4th order Runge Kutta (RK) is employed for time discretization.

Table 4.8: Errors for Example 4: quadratic finite element methods when $B = 0.5$.

#Dof	Δx	$\|\|P_I - P_{\Delta x}\|\|_A$	$\|\|P_I - P_{\Delta x}\|\|_\infty$
959	5.00e-02	9.09381e+00	1.27759e+00
3529	2.50e-02	1.74621e+00	1.24500e-01
13469	1.25e-02	2.61604e-01	1.19905e-02
52549	6.25e-03	3.72530e-02	1.67308e-03
207509	3.13e-03	5.55039e-03	2.31907e-04
Rate		2.7	3

EXAMPLE 4.5 (Moving problems with exact solution)**.** The initial state of the dynamic boundary h is described by $h(x,0) = 1$, and a uniform influx of cells is assumed by $\Psi = -4$. The exact solution are given by $P = 2(y^2 - h^2)$ and $h = e^{-4t}$. Parameters chosen are $K = 1$ and $\xi = 10^{-5}$. We choose the final time $T = 0.004$. Recall that we consider the two dimensional domain $\Omega = (0,2) \times (0,h)$.

$$-K\Delta P = \Psi, x \in \Omega$$

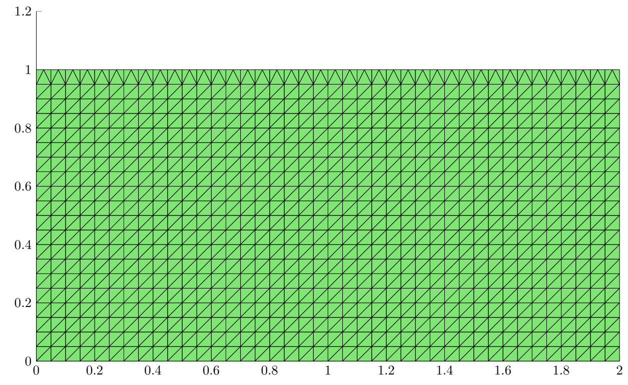
with boundary conditions:

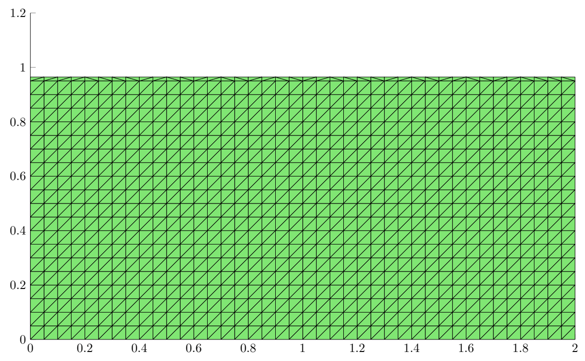
$$v(x, y = 0, t) = \frac{\partial P}{\partial y}(x, y = 0, t) = 0, \quad P(x, h, t) = \xi\kappa = \xi\frac{\partial^2 h/\partial x^2}{[1 + (\partial h/\partial x)^2]^{3/2}},$$

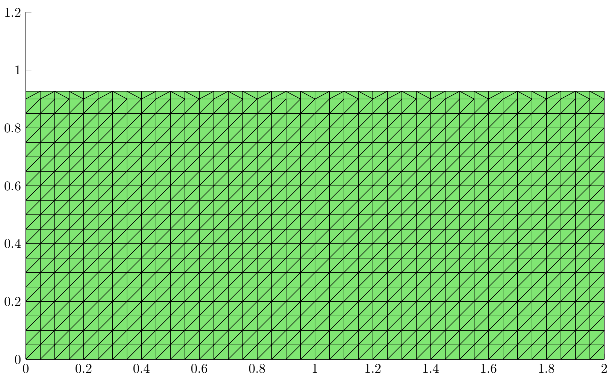
and periodic condition :

$$P(x = 0, y, t) = P(x = 2, y, t).$$

The growth of the dynamic boundary of the system, h is governed by the kinematic condition, $\frac{\partial h}{\partial t} + u(x,h,t)\frac{\partial h}{\partial x} = v(x,h,t)$, where $u = -KP_x$ and $v = -KP_y$. Since the velocity of the transport equation is changing in different time, we choose the time step $\Delta t = \mathcal{O}(\Delta x / \max(u,v))$, where Δx is a uniform span in the x direction. Given the inital boundary condition h, we generate the interface-fitted mesh. The procedure is the same as 4.1.3. Then we use linear finite element methods to solve P. For each boundary node, choose 5 closet neighbour mesh nodes in x and y direction. Fit those points with cubic polynomial using `polyfit`, which is a built-in function in MATLAB. Finally the velocity u and v could be the first derivative of the fitted cubic polynomial. Then we solve the transport equation using DG-FEM to update the new h. The couple system P and h will be solved iteratively. In Figure 4.6, it shows the interface-fitted mesh generated in different time step. From Table 4.9, it shows all the order of spatial accuracy is 2 for P using $||\cdot||_\infty$ and $||\cdot||_A$ norms and for h using $||\cdot||_\infty$ norm.


(a) time = 0.00


(b) time = 0.01


(c) time = 0.02

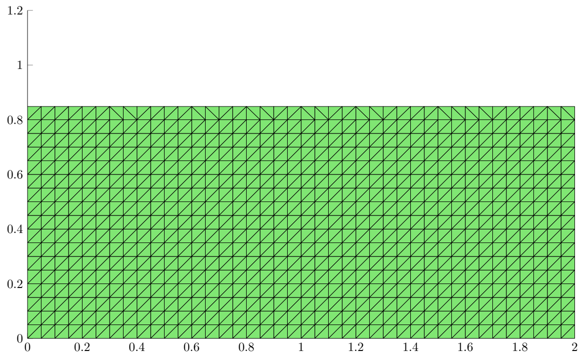

(d) time = 0.04

Figure 4.6: The interface-fitted mesh in the domain with different time when Δx is 0.05.

EXAMPLE 4.6 (Moving problems without exact solution(a straight line))**.** The initial state of the dynamic boundary h is described by $h(x, 0) = 1$ and a uniform influx of cells is assumed by $\Psi = 4$. Parameters chosen are $K = 1$ and $\xi = 10^{-5}$. We choose the final time $T = 0.001$. Recall that we also choose $\Delta t = \mathcal{O}(\Delta x / \max(u, v))$ for time discretization. But the error formulae is different from the above definition because there is not exact solution. Here we use maximum

Table 4.9: Errors for straight line:Errors for P and h.

#Dof	Δx	$\|\|P_I - P_{\Delta x}\|\|_\infty$	$\|\|P_I - P_{\Delta x}\|\|_A$	$\|\|h_I - h_{\Delta x}\|\|_\infty$
231	1.00e-01	3.03654e-03	9.18840e-03	5.72428e-06
861	5.00e-02	7.03826e-04	2.34927e-03	1.06895e-06
3,321	2.50e-02	1.30030e-04	3.83277e-04	2.43114e-07
12,880	1.25e-02	4.78007e-05	3.28391e-04	5.49985e-08
Rate		1.9	1.7	2.1

norm difference between successive approximations to compute the order of accuracy. That is, $\log_2 \frac{\max\left|P_{\Delta x}-P_{\Delta x/2}\right|}{\max\left|P_{\Delta x/2}-P_{\Delta x/4}\right|}$ is the order for P in maximum norm. Similarly, we choose h instead of P to get the convergence rate. Here we need to calculate the approximate curvature since we do not know the exact solution. The idea is the same as computing the velocity. We use cubic polynomial to fit the curve and then take the first and second derivative in the x direction. Table 4.10 demonstrates that the order of spatial accuracy in maximum norm for both P and h is 2.

Table 4.10: Errors for straight line $h = e^{4t}$: Order of accuracy with maximum norm for P and h.

#Dof	Δx	$\|\|P_I - P_{\Delta x}\|\|_\infty$	$\|\|h_I - h_{\Delta x}\|\|_\infty$
544	0.25	2.0748	2.7105
2,112	0.125	2.0155	2.6449
8,320	0.0625	1.9553	2.3873
33,024	0.03125	1.9603	1.7226

4.2 Tumor Growth Problem

4.2.1 Tumor Growth Model

In this section, we apply our mesh algorithm to simulate tumor growth phenomenon. Based on the biological reality, we assume that the tumor occupies the region Ω^- and the healthy tissue stays the outside $\Omega^+ = \Omega\backslash(\Omega^- \cup \Gamma)$ with a free boundary Γ. The pressure and velocity of Ω^+ are simply assumed to be zero. In Ω^-, the following equations govern the tumor growth: Find $(\boldsymbol{u}, p)$ such that

$$\boldsymbol{u} = \nabla p \quad \text{in} \quad \Omega^-, \tag{4.9}$$

$$-\nabla \cdot \boldsymbol{u} = G(n - A) \quad \text{in} \quad \Omega^-, \tag{4.10}$$

$$p\,|_\Gamma = \kappa, \tag{4.11}$$

here $n(\boldsymbol{x}, t)$ is the nutrient (e.g. oxygen) concentration which satisfies

$$\nabla^2 n - n = 0 \quad \text{in} \quad \Omega^-,$$

$$n\,|_\Gamma = 1,$$

$\boldsymbol{u}(\boldsymbol{x}, t)$ is the velocity of the tumor, $p(\boldsymbol{x}, t)$ is the pressure, G and A are respectively the growth and death rates without unit, and κ is the mean curvature on the boundary Γ [143]. To simulate the tumor growth, we need to solve the following elliptic equation:

$$-\Delta p = G(n - A) \quad \text{in} \quad \Omega^-,$$

$$p\,|_\Gamma = \kappa,$$

The interface Γ is represented by a zero level-set function ϕ, that is, $\Gamma(t) = \{\boldsymbol{x} \mid \phi(\boldsymbol{x}, t) = 0\}$. If $\phi < 0$, it is inside Ω^-. If $\phi > 0$, it is outside Ω^+. Suppose the tumor boundary is moving with velocity $\boldsymbol{u} = \dot{\boldsymbol{x}}(t) = \nabla p$. Then we have $\phi(\boldsymbol{x}(t), t) = 0$ by definition. Taking the derivative with respect to time t on both sides, we get the following equation:

$$\frac{\partial \phi}{\partial t} + \boldsymbol{u} \cdot \nabla \phi = 0. \tag{4.12}$$

4.2.2 Numerical Scheme

The mesh generation step is the same as that for the tissue growth problem described in Section 4.1. Then we get the mesh in Ω^- for the initial boundary condition of ϕ. We could directly use the boundary points on Γ to caculate the curvature using the way of our interface-fitted mesh. For each point on Γ, we choose 15 nearby mesh nodes to approximate a circle. The radius of the fitted circle is R. Here the curvature is signed. If one point in the circle is in Ω^-, then the curvature is $1/R$. Otherwise, the curvature is $-1/R$, which is negative. n and p could be solved using linear finite element methods. The gradient of p is obtained using `gradu.m` and `recovery.m` based on the MATLAB® package iFEM [25]. ϕ is defined over the whole domain Ω. But the velocity $\boldsymbol{u} = (p_x, p_y)$ is only computed in Ω^-. Then we extend each component of $\boldsymbol{u}$ from Γ to Ω^+ using harmonic extension. After this, we use WENO5 [115] and 3rd order Runge Kutta methods to solve the transport equation to update ϕ for each time step $\Delta t = \mathcal{O}(\Delta x / \max \boldsymbol{u})$. Then we apply reinitialization step near the interface [104, 141]. The numerical scheme could be summarized as follows:

Step 1: Generate a Cartesian mesh with size h. Based on the initial boundary condition of ϕ, we get the interface-fitted mesh generated by Algorithm 1 in Chapter 2.

Step 2: Compute p, p_x, p_y, κ in Ω^-.

Step 3: Extend p_x, p_y from Γ to Ω^+ using harmonic extension with Dirichlet boundary condition.

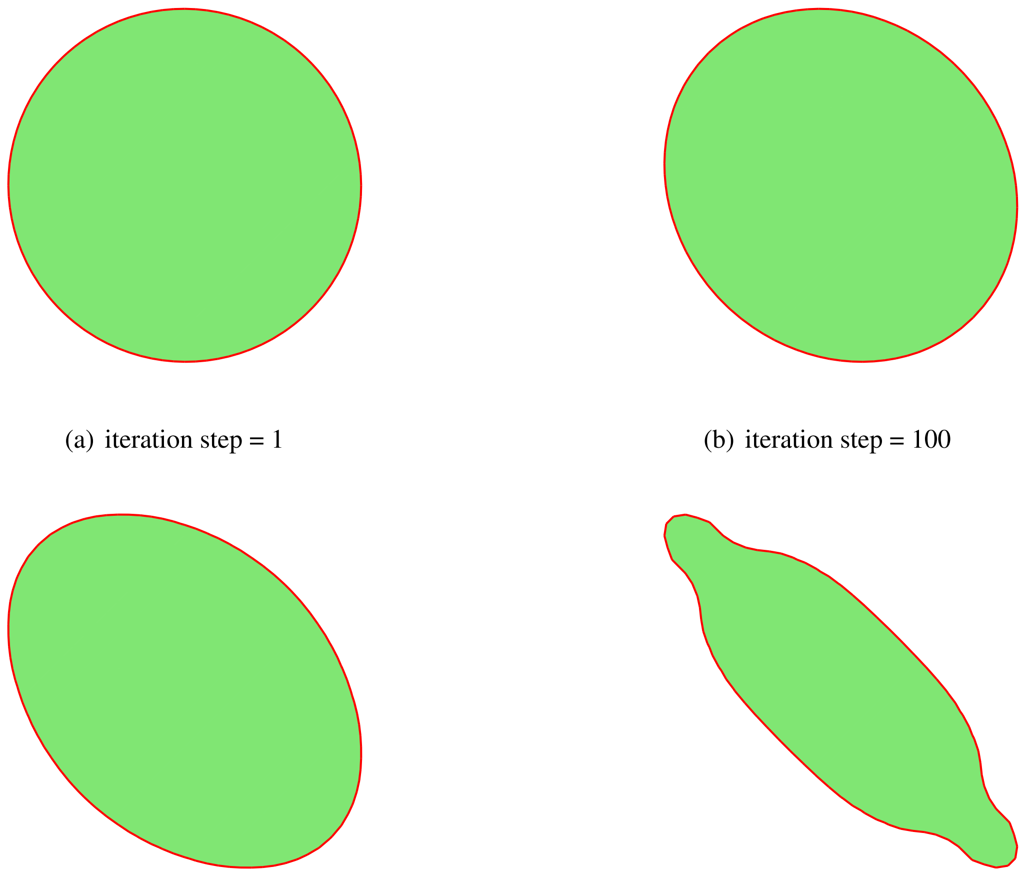
Step 4: Solve the transport equation (4.12) on the Cartesian mesh in Step 1 to update ϕ using WENO5 methods.

Step 5: Reinitialize ϕ near the interface to make it the sign distance function to Γ.

Step 6: Do Step 1 to Step 5 iteratively.

4.2.3 Tumor Growth Simulations

We compute the growth when $G = 10$, $A = 0.5$, and the initial shape of the tumor boundary is $(2\cos(\theta), 2\sin(\theta), \theta \in [0, 2\pi]$. The computational domain is chosen as $\Omega = [-6, 6]^2$. The evolution of a tumor is presented in Figure 4.7.



(a) iteration step = 1

(b) iteration step = 100

(c) iteration step = 300

(d) iteration step = 500

Figure 4.7: The interface(red) and the domain Ω^-(green) when the Cartesian mesh size h is 0.125.

Bibliography

[1] G. Acosta and R. G. Durán. The maximum angle condition for mixed and nonconforming elements: Application to the stokes equations. *SIAM Journal on Numerical Analysis*, 37(1):18–36, 1999.

[2] L. Adams and Z. Li. The immersed interface/multigrid methods for interface problems. *SIAM Journal on Scientific Computing*, 24(2):463–479, 2002.

[3] Al Shenk. Uniform error estimates for certain narrow Lagrangian finite elements. *Math. Comp.*, 63(207):105–119, 1994.

[4] L. Antiga, J. Peiró, and D. A. Steinman. From image data to computational domains. In *Cardiovascular Mathematics*, pages 123–175. Springer, 2009.

[5] I. Babuška. The Finite Element Method for Elliptic Equations with Discontinuous Coefficients. *Computing*, 5(3):207–213, 1970.

[6] I. Babuška and A. K. Aziz. On the angle condition in the finite element method. *SIAM Journal on Numerical Analysis*, 13(2):214–226, 1976.

[7] E. Bänsch, F. Haußer, O. Lakkis, B. Li, and A. Voigt. Finite element method for epitaxial growth with attachment–detachment kinetics. *Journal of Computational Physics*, 194(2):409–434, 2004.

[8] J. W. Barrett and C. M. Elliott. Fitted and Unfitted Finite-Element Methods for Elliptic Equatiosn with Smooth Interfaces. *IMA Journal of Numerical Analysis*, 7:283–300, 1987.

[9] R. Becker, E. Burman, and P. Hansbo. A Nitsche extended finite element method for incompressible elasticity with discontinuous modulus of elasticity. *Computer Methods in Applied Mechanics and Engineering*, 198(41-44):3352–3360, 2009.

[10] L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L. Marini, and A. Russo. Basic principles of virtual element methods. *Mathematical Models and Methods in Applied Sciences*, 23(01):199–214, 2013.

[11] L. Beirao da Veiga, F. Brezzi, L. Marini, and A. Russo. The hitchhiker's guide to the virtual element method. *Mathematical Models and Methods in Applied Sciences*, 24(08):1541–1573, 2014.

[12] B. Bejanov, J.-L. Guermond, and P. D. Minev. A grid-alignment finite element technique for incompressible multicomponent flows. *Journal of Computational Physics*, 227(13):6473–6489, 2008.

[13] P. A. Berthelsen. A decomposed immersed interface method for variable coefficient elliptic equations with non-smooth and discontinuous solutions. *Journal of Computational Physics*, 197(1):364–386, 2004.

[14] C. Börgers. A triangulation algorithm for fast elliptic solvers based on domain imbedding. *SIAM journal on numerical analysis*, 27(5):1187–1196, 1990.

[15] P. Bourke. Calculating the area and centroid of a polygon. 1988.

[16] J. H. Bramble and J. T. King. A finite element method for interface problems in domains with smooth boundaries and interfaces. *Advances in Computational Mathematics*, 6(1):109–138, 1996.

[17] J. Brandts, S. Korotov, M. Křížek, et al. A geometric toolbox for tetrahedral finite element partitions. *Efficient preconditioned solution methods for elliptic partial differential equations*, pages 103–122, 2011.

[18] K. Q. Brown. Voronoi diagrams from convex hulls. *Information Processing Letters*, 9(5):223–228, 1979.

[19] E. Burman, S. Claus, P. Hansbo, M. G. Larson, and A. Massing. CutFEM : Discretizing geometry and partial differential equations. *International Journal for Numerical Methods in Engineering*, 104(December 2014):472–501, 2015.

[20] E. Burman and P. Hansbo. Fictitious domain finite element methods using cut elements: I. A stabilized Lagrange multiplier method. *Computer Methods in Applied Mechanics and Engineering*, 199(41-44):2680–2686, 2010.

[21] E. Burman and P. Hansbo. Fictitious domain finite element methods using cut elements: II. A stabilized Nitsche method. *Applied Numerical Mathematics*, 62(4):328–341, 2012.

[22] Y. C. Chang, T. Y. Hou, B. Merriman, and S. Osher. A Level Set Formulation of Eulerian Interface Capturing Methods for Incompressible Fluid Flows. *Journal of Computational Physics*, 124(2):449–464, Mar. 1996.

[23] D. Chen, Z. Chen, C. Chen, W. Geng, and G.-W. Wei. Mibpb: a software package for electrostatic analysis. *Journal of computational chemistry*, 32(4):756–770, 2011.

[24] L. Chen. Superconvergence of tetrahedral linear finite elements. *International Journal of Numerical Analysis and Modeling*, 3(3):273–282, 2006.

[25] L. Chen. ifem: an integrated finite element methods package in matlab. *University of California at Irvine*, 2009.

[26] L. Chen. Programming of finite element methods in matlab. *arXiv preprint arXiv:1804.05156*, 2018.

[27] L. Chen, H. Wei, and M. Wen. An interface-fitted mesh generator and virtual element methods for elliptic interface problems. *Journal of Computational Physics*, 334:327–348, 2017.

[28] L. Chen and J. Xu. Optimal Delaunay triangulations. *Journal of Computational Mathematics*, 22(2):299–308, 2004.

[29] Z. Chen and J. Zou. Finite element methods and their convergence for elliptic and parabolic interface problems. *Numerische Mathematik*, 79(2):175–202, 1998.

[30] S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng. Silver exudation. *J. ACM*, 47(5):883–904, Sept. 2000.

[31] L. P. Chew. Guaranteed-quality delaunay meshing in 3d (short version). In *Proceedings of the thirteenth annual symposium on Computational geometry*, pages 391–393. ACM, 1997.

[32] M. H. Cho, H. G. Choi, and J. Y. Yoo. A direct reinitialization approach of level-set/splitting finite element method for simulating incompressible two-phase flows. *International Journal for Numerical Methods in Fluids*, 67(11):1637–1654, Dec. 2011.

[33] C. C. Chu, I. G. Graham, and T. Y. Hou. A new multiscale finite element method for high-contrast elliptic interface problems. *Math. Comp*, 79:1915–1955, 2010.

[34] M. Cruchaga, D. Celentano, and T. Tezduyar. A moving Lagrangian interface technique for flow computations over fixed meshes. *Computer Methods in Applied Mechanics and Engineering*, 191(6-7):525–543, Dec. 2001.

[35] F. Dassi, S. Perotto, L. Formaggia, and P. Ruffo. Efficient geometric reconstruction of complex geological structures. *Mathematics and Computers in Simulation*, 106:163–184, 2014.

[36] T. S. Developers. *SageMath, the Sage Mathematics Software System (Version 7.2)*, 2016. `http://www.sagemath.org`.

[37] R. G. Duran. Error estimates for 3–D narrow finite elements. *Math. Comp.*, 68(225):187–199, 1999.

[38] R. G. Durán and A. L. Lombardi. Error estimates for the Raviart-Thomas interpolation under the maximum angle condition. *SIAM Journal on Numerical Analysis*, 46(3):1442–1453, 2008.

[39] H. Edelsbrunner. Triangulations and meshes in computational geometry. *Acta Numerica 2000*, 9:133–213, jan 2000.

[40] H. Edelsbrunner. Triangulations and meshes in computational geometry. *Acta Numerica 2000*, 9:133–213, 2000.

[41] H. Edelsbrunner, X.-Y. Li, G. Miller, A. Stathopoulos, D. Talmor, S.-H. Teng, A. Üngör, and N. Walkington. Smoothing and cleaning up slivers. In *Proceedings of the* 32^{nd} *Annual ACM Symposium on the Theory of Computing*, pages 273–277. ACM, 2000.

[42] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1(1):25–44, 1986.

[43] T.-P. Fries and T. Belytschko. The intrinsic xfem: a method for arbitrary discontinuities without additional unknowns. *International journal for numerical methods in engineering*, 68(13):1358–1385, 2006.

[44] T.-P. Fries and T. Belytschko. The extended/generalized finite element method: An overview of the method and its applications. *International Journal for Numerical Methods in Engineering*, 84(April):253–304, 2010.

[45] P. Frolkovič. Flux-based method of characteristics for contaminant transport in flowing groundwater. *Computing and Visualization in Science*, 5(2):73–83, 2002.

[46] P. Frolkovič and K. Mikula. High-Resolution Flux-Based Level Set Method. *SIAM Journal on Scientific Computing*, 29(2):579–597, Mar. 2007.

[47] S. FUJIMA, Y. FUKASAWA, and M. TABATA. Finite element formulation of periodic conditions and numerical observation of three-dimensional behavior in a flow. 1993.

[48] J. Glimm, J. W. Grove, X. L. Li, and N. Zhao. *Simple front tracking*, volume 238 of *Contemporary Mathematics*. American Mathematical Society, Providence, Rhode Island, 1999.

[49] Y. Gong, B. Li, and Z. Li. Immersed-interface finite-element methods for elliptic interface problems with nonhomogeneous jump conditions. *SIAM Journal on Numerical Analysis*, 46(1):472–495, 2008.

[50] S. Goswami, A. Gillette, and C. Bajaj. Efficient delaunay mesh generation from sampled scalar functions. In *Proceedings of the 16th International Meshing Roundtable*, pages 495–512. Springer, 2008.

[51] G. Guyomarc'h, C.-O. Lee, and K. Jeon. A discontinuous galerkin method for elliptic interface problems with application to electroporation. *Communications in numerical methods in engineering*, 25(10):991–1008, 2009.

[52] J. Guzman, M. Sánchez, and M. Sarkis. On the accuracy of finite element approximations to a class of interface problems. *Mathematics of Computation*, 2015.

[53] A. Hansbo and P. Hansbo. An unfitted finite element method, based on Nitsche's method, for elliptic interface problems. *Computer Methods in Applied Mechanics and Engineering*, 191(47-48):5537–5552, 2002.

[54] A. Hansbo and P. Hansbo. A finite element method for the simulation of strong and weak discontinuities in solid mechanics. *Comput. Methods Appl. Mech. Engrg.*, 193:3523–3540, 2004.

[55] P. Hansbo, M. G. Larson, and S. Zahedi. A cut finite element method for a Stokes interface problem. *Applied Numerical Mathematics*, 85:90–114, 2014.

[56] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of fluids*, 1965.

[57] X. He, T. Lin, and Y. Lin. Immersed finite element methods for elliptic interface problems with non-homogeneous jump conditions. *Int. J. Numer. Anal. Model*, 8(2):284–301, 2011.

[58] J. S. Hesthaven and T. Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007.

[59] J. S. Hesthaven and T. Warburton. *Nodal Discontinuous Galerkin Methods*, volume 54 of *Texts in Applied Mathematics*. Springer New York, New York, NY, 2008.

[60] C. W. Hirt and B. D. Nichols. Volume of fluid (VOF) method for the dynamics of free boundaries. *Journal of Computational Physics*, 39(1):201–225, Jan. 1981.

[61] S. Hou and X.-D. Liu. A numerical method for solving variable coefficient elliptic equation with interfaces. *Journal of Computational Physics*, 202(2):411–445, 2005.

[62] S. Hou, P. Song, L. Wang, and H. Zhao. A weak formulation for solving elliptic interface problems without body fitted grid. *Journal of Computational Physics*, 249:80–95, 2013.

[63] T. Y. Hou, Z. Li, S. Osher, and H. Zhao. A hybrid method for moving interface problems with application to the hele–shaw flow. *Journal of Computational Physics*, 134(2):236–252, 1997.

[64] T. Y. Hou, Z. L. Li, S. Osher, and H. K. Zhao. A hybrid method for moving interface problems with application to the Hele-Shaw flow. *Journal of Computational Physics*, 134(2):236–252, 1997.

[65] J. Huang and J. Zou. Some new a priori estimates for second-order elliptic and parabolic interface problems. *Journal of Differential Equations*, 184(2):570–586, 2002.

[66] J. Huang and J. Zou. Uniform A Priori Estimates for Elliptic and Static Maxwell Interface Problems. *Discrete and Continuous Dynamic Systems-Series B*, 7(1):145–170, 2007.

[67] J.-S. Huh and J. A. Sethian. Exact subgrid interface correction schemes for elliptic interface problems. *Proc. Natl. Acad. Sci.*, 105:9874–9879, 2008.

[68] L. N. T. Huynh, N. C. Nguyen, J. Peraire, and B. C. Khoo. A high-order hybridizable discontinuous Galerkin method for elliptic interface problems. *International Journal for Numerical Methods in Engineering*, 93(June 2012):183–200, 2013.

[69] G. Iaccarino and R. Verzicco. Immersed boundary technique for turbulent flow simulations. *Applied Mechanics Reviews*, 56(3):331–347, 2003.

[70] H. Ji and J. Dolbow. On strategies for enforcing interfacial constraints and evaluating jump conditions with the extended finite element method. *International Journal for Numerical Methods in Engineering*, 61(14):2508–2535, 2004.

[71] A. Johansson and M. G. Larson. A high order discontinuous Galerkin Nitsche method for elliptic problems with fictitious boundary. *Numerische Mathematik*, 123:607–628, 2013.

[72] R. Kafafy, T. Lin, Y. Lin, and J. Wang. Three-dimensional immersed finite element methods for electric field simulation in composite materials. *International journal for numerical methods in engineering*, 64(7):940–972, 2005.

[73] B. Khoo, Z. Li, and P. Lin. *Interface Problems and Methods in Biological and Physical Flows*. World Scientific, 2009.

[74] G. Kossioris, C. Makridakis, and P. E. Souganidis. Finite volume schemes for Hamilton–Jacobi equations. *Numerische Mathematik*, 83(3):427–442, 1999.

[75] M. Krizek. On the maximum angle condition for linear tetrahedral elements. *SIAM Journal on Numerical Analysis*, 29(2):513–520, 1992.

[76] M. Krizek. On the maximum angle condition for linear tetrahedral elements. *SIAM journal on numerical analysis*, 29(2):513–520, 1992.

[77] D.-T. Lee and B. J. Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, 1980.

[78] R. J. Leveque and Z. Li. The immersed interface method for elliptic equations with discontinuous coefficients and singular sources. *SIAM Journal on Numerical Analysis*, 31(4):1019–1044, 1994.

[79] R. J. Leveque and Z. Li. Immersed Interface Methods for Stokes Flow with Elastic Boundaries or Surface Tension. *SIAM J. Sci. Comput.*, 18(3):709–735, 1997.

[80] J. Li, J. M. Melenk, B. Wohlmuth, and J. Zou. Optimal Convergence of Higher Order Finite Element Methods for Elliptic Interface Problems. *Applied Numerical Mathematics*, 60:19–37, 2010.

[81] X.-Y. Li and S.-H. Teng. Generating well-shaped delaunay meshed in 3d. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 28–37, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[82] Z. Li. Immersed interface methods for moving interface problems. *Numerical Algorithms*, 14(4):269–293, 1997.

[83] Z. Li and K. Ito. *The immersed interface method: numerical solutions of PDEs involving interfaces and irregular domains*, volume 33. SIAM, 2006.

[84] Z. Li, T. Lin, and X. Wu. New Cartesian grid methods for interface problems using the finite element formulation. *Numer. Math.*, 96(1):61–98, 2003.

[85] Z. Li, T. Lin, and X. Wu. New cartesian grid methods for interface problems using the finite element formulation. *Numerische Mathematik*, 96(1):61–98, 2003.

[86] T. Lin, Y. Lin, and X. Zhang. Partially penalized immersed finite element methods for elliptic interface problems. *SIAM Journal on Numerical Analysis*, 53(2):1121–1144, 2015.

[87] X.-D. Liu and T. C. Sideris. Convergence of the ghost fluid method for elliptic equations with interfaces. *Math. Comp..*, 72(244):1731–1746, 2003.

[88] R. Löhner, J. R. Cebral, F. E. Camelli, S. Appanaboyina, J. D. Baum, E. L. Mestreau, and O. A. Soto. Adaptive embedded and immersed unstructured grid techniques. *Computer Methods in Applied Mechanics and Engineering*, 197(25):2173–2197, 2008.

[89] P. Macklin and J. S. Lowengrub. A New Ghost Cell/Level Set Method for Moving Boundary Problems: Application to Tumor Growth. *Journal of Scientific Computing*, 35(2-3):266–299, 2008.

[90] E. Marchandise, P. Geuzaine, N. Chevaugeon, and J.-F. Remacle. A stabilized finite element method using a discontinuous level set approach for the computation of bubble dynamics. *Journal of Computational Physics*, 225(1):949–974, July 2007.

[91] E. Marchandise and J.-F. Remacle. A stabilized finite element method using a discontinuous level set approach for solving two phase incompressible flows. *Journal of Computational Physics*, 219(2):780–800, Dec. 2006.

[92] R. Massjung. An Unfitted Discontinuous Galerkin Method Applied to Elliptic Interface Problems. *SIAM J. Numeri. Analysis.*, 50(6):3134–3162, 2012.

[93] J. M. Melenk and I. Babuška. The partition of unity finite element method: basic theory and applications. *Computer methods in applied mechanics and engineering*, 139(1):289–314, 1996.

[94] R. Mittal and G. Iaccarino. IMMERSED BOUNDARY METHODS. *dx.doi.org*, 37(1):239–261, Jan. 2005.

[95] N. Moes, J. Dolbow, and T. Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 150(February):131–150, 1999.

[96] N. Moës, J. Dolbow, and T. Belytschko. A finite element method for crack growth without remeshing. *Int. J. Numer. Meth. Eng*, 46(1):131–150, 1999.

[97] R. Moore and S. Saigal. Eliminating slivers in three-dimensional finite element models. *CMES: Computer Modeling In Engineering & Sciences*, 7(3):283–291, 2005.

[98] L. Mu, J. Wang, X. Ye, and S. Zhao. A new weak Galerkin finite element method for elliptic interface problems. *Journal of Computational Physics*, 325:157–173, 2016.

[99] J. Nitsche. Uber ein Yariationsprinzip zur Liisung von Dirichlet-Problemen bei Verwendung von Teilr˜iumen, die keinen Randbedingungen unterworfen sind. *Abh. Math. Sem. Univ. Hamburg*, 36(2):9–15, 1971.

[100] E. Olsson, G. Kreiss, and S. Zahedi. A conservative level set method for two phase flow II. *Journal of Computational Physics*, 225(1):785–807, July 2007.

[101] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, Nov. 1988.

[102] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.

[103] J. Ovadia and Q. Nie. Numerical methods for two-dimensional stem cell tissue growth. *Journal of scientific computing*, 58(1):149–175, 2014.

[104] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang. A pde-based fast local level set method. *Journal of computational physics*, 155(2):410–438, 1999.

[105] P.-O. Persson and G. Strang. A simple mesh generator in matlab. *SIAM review*, 46(2):329–345, 2004.

[106] C. S. Peskin. Numerical analysis of blood flow in the heart. *Journal of computational physics*, 25(3):220–252, 1977.

[107] C. S. Peskin. The immersed boundary method. *Acta numerica*, 11:479–517, 2002.

[108] C. S. Peskin. The immersed boundary method. *Acta Numer.*, 11:479–517, 2002.

[109] C. Pflaum. Subdivision of boundary cells in 3d, 2000.

[110] C. Pflaum. Semi-unstructured grids. *computing*, 67(2):141–166, 2001.

[111] S. B. Pillapakkam and P. Singh. A Level-Set Method for Computing Solutions to Viscoelastic Two-Phase Flow. *Journal of Computational Physics*, 174(2):552–578, Dec. 2001.

[112] J. Qian, Y.-T. Zhang, and H.-K. Zhao. Fast sweeping methods for eikonal equations on triangular meshes. *SIAM Journal on Numerical Analysis*, 45(1):83–107, 2007.

[113] R. Rangarajan and A. J. Lew. Universal meshes: A new paradigm for computing with nonconforming triangulations. *arXiv preprint arXiv:1201.4903*, 2012.

[114] J. A. Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, volume 3. Cambridge university press, 1999.

[115] C. Shu. WENO methods. *Scholarpedia*, 6(5):9709, 2011. revision #91939.

[116] A. Smolianski. Finite-element/level-set/operator-splitting (FELSOS) approach for computing two-fluid unsteady flows with free moving interfaces. *International Journal for Numerical Methods in Fluids*, 48(3):231–269, May 2005.

[117] J. Strain. Tree Methods for Moving Interfaces. *Journal of Computational Physics*, 151(2):616–648, May 1999.

[118] G. Strang and G. J. Fix. *An analysis of the finite element method*, volume 212. Prentice-Hall Englewood Cliffs, NJ, 1973.

[119] M. Sussman and E. Fatemi. An Efficient, Interface-Preserving Level Set Redistancing Algorithm and Its Application to Interfacial Incompressible Fluid Flow. *SIAM Journal on Scientific Computing*, 20(4):1165–1191, July 2006.

[120] M. Sussman, P. Smereka, and S. Osher. A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow. *Journal of Computational Physics*, 114(1):146–159, Sept. 1994.

[121] O. J. Sutton. The virtual element method in 50 lines of matlab. *Numerical Algorithms*, 75(4):1141–1159, 2017.

[122] C. Talischi, G. H. Paulino, A. Pereira, and I. F. Menezes. Polymesher: a general-purpose mesh generator for polygonal elements written in matlab. *Structural and Multidisciplinary Optimization*, 45(3):309–328, 2012.

[123] A.-K. Tornberg and B. Engquist. A finite element based level-set method for multiphase flow applications. *Computing and Visualization in Science*, 3(1-2):93–101, 2000.

[124] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y. J. Jan. A Front-Tracking Method for the Computations of Multiphase Flow. *Journal of Computational Physics*, 169(2):708–759, May 2001.

[125] S. P. Van der Pijl, A. Segal, C. Vuik, and P. Wesseling. A mass-conserving Level-Set method for modelling of multi-phase flows. *International Journal for Numerical Methods in Fluids*, 47(4):339–361, Feb. 2005.

[126] S. P. Van der Pijl, A. Segal, C. Vuik, and P. Wesseling. Computing three-dimensional two-phase flows with a mass-conserving level set method. *Computing and Visualization in Science*, 11(4-6):221–235, 2008.

[127] E. Wadbro, S. Zahedi, and G. Kreiss. A uniformly well-conditioned, unfitted Nitsche method for interface problems. *BIT Numer Math*, 53:791–820, 2013.

[128] F. Wang, Y. Xiao, and J. Xu. High-Order Extended Finite Element Methods for Solving Interface Problems. *ArXiv*, pages 1–25, 2016.

[129] H. Wei, L. Chen, Y. Huang, and B. Zheng. Adaptive mesh refinement and superconvergence for two-dimensional interface problems. *SIAM Journal on Scientific Computing*, 36(4):A1478–A1499, 2014.

[130] J. E. Welch, F. H. Harlow, and J. P. Shannon. The MAC method, 1966.

[131] A. Wiegmann and K. P. Bube. The explicit-jump immersed interface method: finite difference methods for pdes with piecewise smooth solutions. *SIAM Journal on Numerical Analysis*, 37(3):827–862, 2000.

[132] D. E. Womble. A Front-Tracking Method for Multiphase Free Boundary Problems. *SIAM Journal on Numerical Analysis*, 26(2):380–396, July 2006.

[133] H. Wu and Y. Xiao. An unfitted hp-interface penalty finite element method for elliptic interface problems. *arXiv preprint arXiv:1007.2893*, 2010.

[134] K. Xia, M. Zhan, and G.-W. Wei. MIB Galerkin method for elliptic interface problems. *Journal of Computational and Applied Mathematics*, 272(7):195–220, 2014.

[135] H. Xie, Z. Li, and Z. Qiao. A finite element method for elasticity interface problems with locally modified triangulations. *International journal of numerical analysis and modeling*, 8(2):189, 2011.

[136] J. Xu. Error estimates of the finite element method for the 2nd order elliptic equations with discontinuous coefficients. *J. Xiangtan University*, 1:1–5, 1982.

[137] J. Xu. Estimate of the Convergence Rate of Finite Element Solutions to Elliptic Equations of Second Order with Discontinuous Coefficients. *Natural Science Journal of Xiangtan University*, 1(1):1–5, 1982.

[138] J.-J. Xu and H.-K. Zhao. An Eulerian Formulation for Solving Partial Differential Equations Along a Moving Interface. *Journal of Scientific Computing*, 19(1-3):573–594, 2003.

[139] J. Yang and E. Balaras. An embedded-boundary formulation for large-eddy simulation of turbulent flows interacting with moving boundaries. *Journal of Computational Physics*, 215(1):12–40, June 2006.

[140] S. Yu and G. W. Wei. Three-dimensional matched interface and boundary (MIB) method for treating geometric singularities. *Journal of Computational Physics*, 227(1):602–632, 2007.

[141] K. Zhang, L. Zhang, H. Song, and D. Zhang. Reinitialization-free level set evolution via reaction diffusion. *IEEE Transactions on Image Processing*, 22(1):258–271, 2013.

[142] X. Zhang. *Nonconforming immersed finite element methods for interface problems*. PhD thesis, Virginia Polytechnic Institute and State University, 2013.

[143] X. Zheng and J. Lowengrub. An interface-fitted adaptive mesh method for elliptic problems and its application in free interface problems with surface tension. *Advances in Computational Mathematics*, 42(5):1225–1257, 2016.

[144] Y. Zhou and T. H. North. Kinetic modelling of diffusion-controlled, two-phase moving interface problems. *Modelling and Simulation in Materials Science and Engineering*, 1(4):505–516, July 1993.

[145] Y. Zhou and G.-W. Wei. On the fictitious-domain and interpolation formulations of the matched interface and boundary (mib) method. *Journal of Computational Physics*, 219(1):228–246, 2006.

[146] Y. Zhou, S. Zhao, M. Feig, and G.-W. Wei. High order matched interface and boundary method for elliptic equations with discontinuous coefficients and singular sources. *Journal of Computational Physics*, 213(1):1–30, 2006.

[147] Y. C. Zhou, S. Zhao, M. Feig, and G. W. Wei. High order matched interface and boundary method for elliptic equations with discontinuous coefficients and singular sources. *J. Comput. Phys.*, 213(1):1–30, 2006.

Appendix A

Appendices

A.1 The Elliptic Interface Equation and Weak Formulation

Elliptic interface problems arise in many physical applications, including electromagnetism, fluids dynamics, materials science, free boundary problems, and biological applications [49, 69, 114]. In this work we study a typical elliptic interface problem which is described as piece-wisely defined elliptic partial differential equations (PDE) in different regions which are coupled together with non-homogeneous jump conditions, such as solution and flux jump across the interface.

A.1.1 The Elliptic Interface Equation

Ω is an open bounded domain in $\mathbb{R}^d, d = 2, 3$ and Γ is a continuous simple connected interface which separates the domain Ω into two disjoint regions Ω^+ and Ω^-. Ω^+ denotes the exterior domain and Ω^- denotes the interior domain enclosed by Γ. The equation of the typical elliptic interface problem is as follows:

$$-\nabla \cdot (\beta(x)\nabla u(x)) = f(x), x \in \Omega\backslash\Gamma \tag{A.1}$$

with solution and flux jump conditions across the interface Γ:

$$[u]_\Gamma = u^+ - u^- = q_0, \qquad [\beta u_n]_\Gamma = \beta^+ u_n^+ - \beta^- u_n^- = q_1, \qquad u = g \quad \text{on } \partial\Omega, \tag{A.2}$$

where u_n is the normal flux $(\nabla u) \cdot n$ and n is the unit norm direction of the interface Γ pointing from Ω^- to Ω^+(outwards). The superscripts $+$ and $-$ stand for the constraint of a function on Ω^+ and Ω^-, respectively. The coefficient $\beta(x)$ is assumed to be uniformly positive and smooth on each disjoint subdomain, but may be discontinuous across the interface Γ [27].

A.1.2 The Weak Formulation

Given functions q_0 and q_1 along the interface, we prescribe the non-homogeneous jump conditions in Eq. (A.2). The flux jump q_1 is imposed weakly in the weak formulation. Elliptic interface problems with such jump conditions arise in many application fields. For example, BurtonCabreraFrank-type model [7] solves the diffusion equation on terraces with jump condition across the interface representing atomic steps. Another example is that the reaction potential model of electrostatics of a solvation energy has a non homogeneous flux jump condition [49].

Proposition A.1. *The homogeneous jump conditions are $q_0 = 0$ and $q_1 = 0$ on Γ((A.2)), then the problem* (A.1)-(A.2) *is to find $u \in H^1(\Omega)$ with $u = g$ on $\partial\Omega$ such that*

$$\int_\Omega \beta\nabla u \cdot \nabla v \, dx = \int_\Omega fv dx, \quad \forall v \in H_0^1(\Omega)$$

Proof. Multiply the equation by a smooth test function $v \in H_0^1(\Omega)$ and integrate over the whole domain Ω,

$$\int_{\Omega} -\nabla \cdot (\beta \nabla u) v dx = \int_{\Omega} f v dx, \forall v \in H_0^1(\Omega) \tag{A.3}$$

then use integration by parts

$$\int_{\Omega} -\nabla \cdot (\beta \nabla u) v dx = -\int_{\partial\Omega} \beta \nabla u \cdot n v ds + \int_{\Omega} \beta \nabla u \cdot \nabla v dx = \int_{\Omega} \beta \nabla u \cdot \nabla v dx \tag{A.4}$$

Thus combining (A.3) and (A.4), we get

$$\int_{\Omega} \beta \nabla u \cdot \nabla v dx = \int_{\Omega} f v dx, \forall v \in H_0^1(\Omega)$$

Note that $\int_{\partial\Omega} \beta \nabla u \cdot n v ds = 0$, since $v = 0$ on $\partial\Omega$ and $q_1 = 0$ on Γ. □

Proposition A.2. *If $q_0 = 0$, $q_1 \neq 0$ on Γ, then the corresponding weak formulation is :*

$$(\beta \nabla u, \nabla v)_{\Omega} = (f, v)_{\Omega} - \langle q_1, v \rangle_{\Gamma}, \quad \forall v \in H_0^1(\Omega) \tag{A.5}$$

Proof. From (A.1), we could write in this way:

$$-\nabla \cdot (\beta^-(x) \nabla u(x)) = f(x), x \in \Omega^-, \quad -\nabla \cdot (\beta^+(x) \nabla u(x)) = f(x), x \in \Omega^+$$

For the first equation, using integration by parts,

$$\int_{\partial\Omega^-} -\beta^- \triangle u \cdot n v ds + \int_{\Omega^-} \beta^- \nabla u \cdot \nabla v dx = \int_{\Omega^-} f v dx$$

for all $v \in H_0^1(\Omega)$. Since $\partial\Omega^- = \Gamma$, then

$$\int_{\Gamma} -\beta^- \triangle u \cdot n v ds + \int_{\Omega^-} \beta^- \nabla u \cdot \nabla v dx = \int_{\Omega^-} f v dx. \tag{A.6}$$

Similarly, for the second equation, using integration by parts,

$$\int_{\partial\Omega^+} -\beta^+ \triangle u \cdot \tilde{n} v ds + \int_{\Omega^+} \beta^+ \nabla u \cdot \nabla v dx = \int_{\Omega^+} f v dx$$

for all $v \in H_0^1(\Omega)$. Since $\partial\Omega^+ = \partial\Omega \cup \Gamma$, $v = 0$ on $\partial\Omega$ and $\tilde{n} = -n$, then

$$\int_{\Gamma} \beta^+ \triangle u \cdot n v ds + \int_{\Omega^+} \beta^+ \nabla u \cdot \nabla v dx = \int_{\Omega^+} f v dx. \tag{A.7}$$

Adding (A.6) and (A.7), we could get

$$\int_{\Gamma} (\beta^+ \triangle u^+ - \beta^- \triangle u^-) \cdot n v ds + \int_{\Omega} \beta \nabla u \cdot \nabla v dx = (f, v)_{\Omega}, \quad \forall v \in H_0^1(\Omega)$$

Since $q_1 = (\beta^+ \triangle u^+ - \beta^- \triangle u^-) \cdot n$, then

$$\int_{\Gamma} q_1 v ds + \int_{\Omega} \beta \nabla u \cdot \nabla v dx = (f, v)_{\Omega}, \quad \forall v \in H_0^1(\Omega)$$

Removing the first term to the right hand side, then

$$(\beta \nabla u, \nabla v)_{\Omega} = (f, v)_{\Omega} - \langle q_1, v \rangle_{\Gamma}, \quad \forall v \in H_0^1(\Omega)$$

On the other direction, if we choose $v \in H_0^1(\Omega^+)$,then we get

$$\int_{\Omega} \beta^+ \nabla u \cdot \nabla v dx = \int_{\Omega^+} f v dx, \forall v \in H_0^1(\Omega^+).$$

Similarly, if we choose $v \in H_0^1(\Omega^-)$, then we get

$$\int_{\Omega} \beta^- \nabla u \cdot \nabla v dx = \int_{\Omega^-} f v dx, \forall v \in H_0^1(\Omega^-)$$

Finally, we get $\int_{\Gamma} (\beta^+ \triangle u^+ - \beta^- \triangle u^-) \cdot n v ds = \int_{\Gamma} q_1 v ds$ when choosing $v \in H^{\frac{1}{2}}(\Gamma)$. □

Proposition A.3. *If $q_0 \neq 0$ on Γ, we can find $w^- : \Omega^- \to R$ with $w^- = q_0$ on $\partial\Omega^-$ and $w^- \in H^1(\Omega^-)$. The zero extension of w^- ,denote by w satisfies $w \in H^1(\Omega^- \cup \Omega^+)$ with $w = w^-$ on Ω^- and $w = 0$ on Ω^+. The model is equivalent to: find $u = p - w$ with $p \in H_g^1(\Omega)$ such that:*

$$(\beta\nabla p, \nabla v)_\Omega = (f, v)_\Omega - \langle q_1, v\rangle_\Gamma + (\beta\nabla w, \nabla v)_{\Omega^-}, \quad \forall v \in H_0^1(\Omega) \tag{A.8}$$

Proof. Since p satisfies $[p]_\Gamma = 0$ and $[\beta p_n]_\Gamma = q_1 + [\beta w_n]_\Gamma$, then (A.5) holds for p, that is

$$(\beta\nabla p, \nabla v)_\Omega = (f - \nabla \cdot (\beta\nabla w), v)_\Omega - \langle q_1 + [\beta w_n]_\Gamma, v\rangle_\Gamma, \quad \forall v \in H_0^1(\Omega)$$

then

$$(\beta\nabla p, \nabla v)_\Omega = (f, v)_\Omega - \langle q_1, v\rangle_\Gamma - (\nabla \cdot (\beta\nabla w), v)_\Omega - \langle [\beta w_n]_\Gamma, v\rangle_\Gamma, \quad \forall v \in H_0^1(\Omega)$$

Since $\forall v \in H_0^1(\Omega)$

$$-(\nabla \cdot (\beta\nabla w), v)_\Omega = -(\nabla \cdot (\beta\nabla w), v)_{\Omega^-} = -\int_\Gamma \beta\nabla w \cdot n \cdot v ds + \int_{\Omega^-} \beta\nabla u \cdot \nabla v dx,$$

and

$$\langle [\beta w_n]_\Gamma, v\rangle_\Gamma = -\int_\Gamma \beta\nabla w \cdot n \cdot v ds,$$

then we get

$$(\beta\nabla p, \nabla v)_\Omega = (f, v)_\Omega - \langle q_1, v\rangle_\Gamma + (\beta\nabla w, \nabla v)_{\Omega^-}, \quad \forall v \in H_0^1(\Omega)$$

□

THEOREM A.4. *The weak formulation of the elliptic interface equation ((A.1)-(A.2)) is stated as: find $u = p - w$ with $p \in H^1_g(\Omega) = \{v \in H^1(\Omega) : v|_{\partial\Omega} = g\}$ such that*

$$(\beta \nabla p, \nabla v)_\Omega = (f, v)_\Omega - \langle q_1, v\rangle_\Gamma + (\beta \nabla w^-, \nabla v)_{\Omega^-}, \quad \forall v \in H^1_0(\Omega). \tag{A.9}$$

A.2 Efficient Implementation of Virtual Element Methods in Two Dimensions

In this section, we mainly present how to implement the linear virtual element method for the two dimensional Poisson equation on a polygonal domain. In [121], it aims to short implementation of algorithms for the education purpose. But the codes could be more efficient especially for the large size of matrix. Here we focus on efficient implementation in Matlab.

A.2.1 Introduction

Virtual Element Method (VEM) [10] is developed by Brezzi's group, which can be viewed as an extension of Finite Element Methods to general polygonal and polyhedral elements. In two dimensions, elements in finite element methods are either triangles or quadrilateral. But in VEM, polygonal elements can be of very general shape, which may even be non convex. Theoretical results and generalize matrix formulation are discussed in [10, 11]. [11]is about the coding. Here we mainly focus on the efficient implementation of linear virtual element methods. At first, we will introduce the concrete matrix formulation for the linear virtual element methods. Then we will demonstrate how to implement each component of the matrix in the next section.

First we study the model problem of the two dimensional Poisson equation:

$$\begin{cases} -\Delta u = f \text{ in } \Omega, \\ u = g \text{ on } \partial\Omega. \end{cases} \tag{A.10}$$

Here Ω is a polygonal domain in $\mathbb{R}^2$, $f \in L^2(\Omega)$ and $g \in H^{1/2}(\partial\Omega)$. Define $H_0^1(\Omega) := \{v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega\}$ and $H_g^1(\Omega) := \{v \in H^1(\Omega) : v = g \text{ on } \partial\Omega\}$. The weak formulation will be derived using the same strategy of the classical finite element methods: find $u \in H_g^1(\Omega)$ such that

$$a(u, v) = L(v), \quad \forall v \in H_0^1(\Omega),$$

where the bilinear form $a(u, v) = (\nabla u, \nabla v) = \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x$, the linear form $L(v) = (f, v) = \int_\Omega f v \, \mathrm{d}x$ and $(\cdot, \cdot)$ is the inner product in L^2. By Lax-Milgram Lemma, it has a unique solution.

Let $\mathcal{T}_h$ be the collection of partitions of the domain Ω into polygonal elements with not self-intersecting boundary. h denotes the maximum diameter of all elements in $\mathcal{T}_h$. In our numerical results, h is approximated by the square root of the area of each element. A local finite-dimensional vector space $V_h(E)$ for a polygon $E \in \mathcal{T}_h$ can be written as

$$V_h(E) := \{v \in H^1(E) : \Delta v|_E = 0, v|_{\partial E} \text{ is continuous and piecewise linear polynomial}\}.$$

Since a piecewise linear function will be uniquely determined by its value on vertices, $\dim V_h(E) = n_E^v$, where n_E^v is the number of vertices of E.

The global virtual element space is defined as

$$V_h = \{v_h \in H^1(\Omega) : v_h|_E \in V_h(E) \text{ for all } E \in \mathcal{T}_h\}.$$

$\mathcal{N}(\mathcal{T}_h)$ is the set of vertices of mesh $\mathcal{T}_h$ and $N = |\mathcal{N}(\mathcal{T}_h)|$ denotes the number of vertices. We define the operator dof_i from V_h to $\mathbb{R}$ as $\mathrm{dof}_i(v_h) = v_h(\boldsymbol{x}_i)$, for a vertex $\boldsymbol{x}_i \in \mathcal{N}(\mathcal{T}_h)$. The canonical basis $\{\phi_1, \cdots, \phi_N\} \subset V_h$ is chosen as $\mathrm{dof}_i(\phi_j) = \delta_{ij}, i, j = 1, \cdots, N$. And the nodal interpolation $I_h : C(\bar{\Omega}) \to V_h$ is defined as $I_h u = \sum_{i=1}^N u(\boldsymbol{x}_i)\phi_i$ and denoted by $u_I = I_h u$.

In finite element methods, we use discrete solutions u_h to approximate the solution u. The linear virtual element approximation of (A.10) is: finding $u_h \in V_h \cap H_g^1(\Omega)$ such that:

$$(\nabla u_h, \nabla v_h)_\Omega = (f, v_h)_\Omega, \quad \forall v_h \in V_h \cap H_0^1(\Omega).$$

Suppose $u_h = \sum_{j=1}^N u_j \phi_j$, by linearity, we have for $i \in 1, \cdots, N$,

$$\sum_{j=1}^{N} (\nabla \phi_j, \nabla \phi_i)_\Omega u_j = (f, \phi_i)_\Omega. \tag{A.11}$$

We denote the matrix $\boldsymbol{A}_{ij} = (\nabla \phi_j, \nabla \phi_i)_{\mathcal{T}_h}$ and the vector $\boldsymbol{b} = (b_1, \cdots, b_N)^t$ by $b_i = (f, \phi_i)_\Omega$. Equation (A.11) is written in the matrix form as

$$\boldsymbol{A}_h \boldsymbol{u}_h = \boldsymbol{b}. \tag{A.12}$$

Here $\boldsymbol{A}_h$ is an $N \times N$ matrix and $\boldsymbol{u}_h = (u_h^1, \cdots, u_h^N)^t$. As the matrix $\boldsymbol{A}_h$ is symmetric and positive definite, the algebraic system (A.12) could be solved stably and efficiently by using algebraic multigrid methods.

For finite element methods, it suffices to compute the local stiffness matrix in each element at first, and then the matrix $\boldsymbol{A}$ is assembled by summing the contribution from each element. Therefore, the major task is to compute $(\nabla \phi_j, \nabla \phi_i)_E$, where $(\cdot, \cdot)_E$ is the L^2 inner product on E.

In order to do so, we define some projection operators at first. In each polygonal element E, the operator $\Pi^\nabla : V_h(E) \to \mathbb{P}_1(E)$ is represented as the H^1 projection to $\mathbb{P}_1(E)$ space, i.e.,:

$$(\nabla p_k, \nabla \Pi^\nabla v_h)_E = (\nabla p_k, \nabla v_h)_E \quad \text{for all } p_k \in \mathbb{P}_1(E).$$

Here $\mathbb{P}_1(E)$ is the space of linear polynomials. It is obvious that the above condition defines $\Pi^\nabla v_h$ only up to a constant. This can be resolved by mapping a projection operator onto constants $P_0 : V_h(E) \to \mathbb{P}_0(E)$ and requiring

$$P_0(\Pi^\nabla v_h - v_h) = 0.$$

One simple choice is $P_0 v_h = \sum_{i=1}^{n_E^v} v_h(\boldsymbol{x}_i)/n_E^v = \sum_{i=1}^{n_E^v} \mathrm{dof}_i(v_h)/n_E^v$.

We rewrite the basis function $\phi_i \in V_h(E)$ as $\Pi^\nabla \phi_i + (I - \Pi^\nabla)\phi_i$ using the projection Π^∇, and then split the entry of the local stiffness matrix as

$$(\nabla \Pi^\nabla \phi_i, \nabla \Pi^\nabla \phi_j)_E + (\nabla(I - \Pi^\nabla)\phi_i, \nabla(I - \Pi^\nabla)\phi_j)_E.$$

Again the second term is not computable since the basis ϕ_i is not known point-wise. Now we replace by a so-called stabilization term $S^E(\cdot,\cdot)$

$$(\nabla \Pi^\nabla \phi_i, \nabla \Pi^\nabla \phi_j)_E + S^E((I - \Pi^\nabla)\phi_i, (I - \Pi^\nabla)\phi_j).$$

A scaled l^2 inner product was chosen in the stabilization term

$$S^E((I - \Pi^\nabla)\phi_i, (I - \Pi^\nabla)\phi_j) = \sum_{r=1}^{n_E^v} \mathrm{dof}_r((I - \Pi^\nabla)\phi_i)\, \mathrm{dof}_r((I - \Pi^\nabla)\phi_j)$$

to satisfy the assumption of S^E

$$c_1(\nabla v, \nabla v) \le S^E(v,v) \le c_2(\nabla v, \nabla v), \quad \forall v \in V_h(E) \text{ and } \Pi^\nabla v = 0,$$

where some positive constants c_1 and c_2 are independent of E [10]. We write the explicit expression of the local stiffness matrix of the virtual element method as follows:

$$(\boldsymbol{A}_h^E)_{ij} := (\nabla\Pi^\nabla\phi_i, \nabla\Pi^\nabla\phi_j)_E + \sum_{r=1}^{n_E^v} \mathrm{dof}_r((I-\Pi^\nabla)\phi_i)\,\mathrm{dof}_r((I-\Pi^\nabla)\phi_j). \tag{A.13}$$

The matrix representation of the operator Π^∇ is computed using the following concrete formulae. $\boldsymbol{x}_E = (x_E, y_E)$ denotes the centroid of E. A scaled monomial basis of $\mathbb{P}_1(E)$ is chosen as $m_1 = 1, m_2 = (x-x_E)/h_E, m_3 = (y-y_E)/h_E$, where $h_E = |E|^{1/2}$, which is approximated by the square root of the area of each element E. Let the matrix $\boldsymbol{G}_{3\times 3}$ be defined as

$$\boldsymbol{G} := \begin{pmatrix} P_0 m_1 & P_0 m_2 & P_0 m_3 \\ 0 & (\nabla m_2, \nabla m_2)_{0,E} & (\nabla m_3, \nabla m_2)_{0,E} \\ 0 & (\nabla m_2, \nabla m_3)_{0,E} & (\nabla m_3, \nabla m_3)_{0,E} \end{pmatrix} = \begin{pmatrix} 1 & \frac{1}{n_E^v}\sum_{i=1}^{n_E^v}\frac{x_i - x_E}{h_E} & \frac{1}{n_E^v}\sum_{i=1}^{n_E^v}\frac{y_i - y_E}{h_E} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Here $\boldsymbol{B}_{3\times n_E^v}$ is a matrix defined as:

$$\boldsymbol{B} := \begin{pmatrix} P_0\phi_1 & \cdots & P_0\phi_{n_E^v} \\ (\nabla m_2, \nabla\phi_1)_E & \cdots & (\nabla m_2, \nabla\phi_{n_E^v})_E \\ (\nabla m_3, \nabla\phi_1)_E & \cdots & (\nabla m_3, \nabla\phi_{n_E^v})_E \end{pmatrix}.$$

The first row of $\boldsymbol{B}$ is $P_0\phi_1 = P_0\phi_2 = \ldots = P_0\phi_{n_E^v} = 1/n_E^v$. For the other components $(\nabla m_j, \nabla\phi_i)_E, j = 2,3$, we have $(\nabla m_j, \nabla\phi_i)_E = -\int_E \Delta m_j \phi_i + \int_{\partial E}\frac{\partial m_j}{\partial n}\phi_i$ by integration by parts. Here the first term is zero as $\Delta m_j = 0$ for linear polynomials and the second term need to be only computed.

$$\int_{\partial E}\frac{\partial m_j}{\partial n}\phi_i = \frac{\sum_{\text{node } i\in\text{edge e}} n_e^j}{2h_E}, \tag{A.14}$$

where $\boldsymbol{n} = (n_e^x, n_e^y) = (n_e^2, n_e^3)$ denotes an outward unit normal direction to the edge e.

In order to compute the stabilization term, one more matrix $\boldsymbol{D}_{n_E^v \times 3}$ is needed.

$$\boldsymbol{D} := \Big(\mathrm{dof}_i(m_j)\Big) = h_E^{-1} \begin{pmatrix} h_E & x_1 - x_E & y_1 - y_E \\ h_E & x_2 - x_E & y_2 - y_E \\ \cdots & \cdots & \cdots \\ h_E & x_{n_E^v} - x_E & y_{n_E^v} - y_E \end{pmatrix},$$

where $(x_i, y_i), i = 1, \cdots, n_E^v$ mean the vertices in each polygon E.

By definition,

$$\Pi^\nabla v_h = \sum_{\alpha=1}^{3} s^\alpha m_\alpha \tag{A.15}$$

and the coefficients (s^α) are determined based on the following linear systems

$$(\nabla m_\alpha, \nabla(\Pi^\nabla v_h - v_h))_E = 0 \quad \alpha = 1, \ldots, 3. \tag{A.16}$$

The matrix representation of $\Pi^\nabla : V_h(E) \to \mathbb{P}_1(E)$ relative to the basis (m_α) is $\boldsymbol{\Pi}^\nabla = \boldsymbol{G}^{-1}\boldsymbol{B}$.

Furthermore, we also need the matrix representation of Π^∇ in the canonical basis $\{\phi_i\}$. Let $\Pi^\nabla \phi_i = \sum_{j=1}^{n_E^v} \mathrm{dof}_j(\Pi^\nabla \phi_i)\phi_j, i = 1, \cdots, n_E^v$, then the matrix representation $\boldsymbol{\Pi}_*^\nabla$ of the operator $\Pi^\nabla : V_h(E) \to V_h(E)$ in the canonical basis is written as $\boldsymbol{\Pi}_*^\nabla = \boldsymbol{D}\boldsymbol{G}^{-1}\boldsymbol{B} = \boldsymbol{D}\boldsymbol{\Pi}^\nabla$.

Finally the matrix formulation of $\boldsymbol{A}_h^E$ could be given by

$$\boldsymbol{A}_h^E = [\boldsymbol{\Pi}^\nabla]^T \tilde{\boldsymbol{G}} \boldsymbol{\Pi}^\nabla + [\boldsymbol{I} - \boldsymbol{\Pi}_*^\nabla]^T [\boldsymbol{I} - \boldsymbol{\Pi}_*^\nabla],$$

where $\tilde{\boldsymbol{G}}$ is the same with $\boldsymbol{G}$ except that the elements in the first row are all zero.Then it is easily to get the local stiffness matrices $(\boldsymbol{A}_h^-|_E)$ and $(\boldsymbol{A}_h^+|_E)$.

For the right hand side vector b_i in (A.12), we simply approximate f by a piecewise constant and approximate

$$(f,\phi_i)_\Omega = \sum_{E\in\mathcal{T}_h}(f,\phi_i)_E \approx \sum_{E\in\mathcal{T}_h}|E|f(x_E,y_E)/n_E^v.$$

A.2.2 Implementation

In this subsection, we discuss how to implement the code efficiently in Matlab. In [121], it shows how to implement the code in Matlab based on the matrix form G, B and D, but it is not efficient for two reasons. One reason is that there are large `for` loops. Vectorization should be taken into account to avoid `for` loop as much as possible. Another one is that it does not take advantage of sparsity. Because the stiffness matrix we obtained via the discretization is very sparse. We shall think about using sparse matrix algorithms, which require less computational time and computer memory [26].

The polygonal mesh we used is generated from `PolyMesher.m`, which is written in MATLAB [122]. The output contains a matrix named `node` which represents the coordinates of vertices and a cell array named `element` which records the vertices of each element with a counter-clockwise order. Given `node` and `element`, the stiffness matrix A could be implemented as follows:

```
function A = assemblematrix(node,element)
N = size(node,1);
Nelement = size(element,1);
% initialize large enough array
ii = zeros(Nelement*49,1);
jj = zeros(Nelement*49,1);
ss = zeros(Nelement*49,1);
curIdx = 1; % current index
for t = 1:Nelement
   v = element{t};
   Nv = length(v);
   x1 = node(v,1);% the coordinate of node
```

```
    y1 = node(v,2);
    x2 = circshift(x1,-1);
    y2 = circshift(y1,-1);
    bdIntegral = x1.*y2 - y1.*x2;
    area = sum(bdIntegral)/2;% the area
    h = sqrt(abs(area));
    cx = sum((x1+x2).*bdIntegral)/(6*area); % the centroid
    cy = sum((y1+y2).*bdIntegral)/(6*area);
    edgeVecx = x2 - x1; %v(k-1)-v(k) for x (-x)
    edgeVecy = y2 - y1; %v(k-1)-v(k) for y (-y)
    normVecx = edgeVecy; % the first normal derivative (-y)
    normVecy = -edgeVecx; % the second normal derivative (x)
    H = zeros(2,Nv);
    H(1,:) = (normVecx + circshift(normVecx,1))/(2*h);
    H(2,:) = (normVecy + circshift(normVecy,1))/(2*h);
    % Part 1: matrix of projections
    A1 = H'*H;
    % Part 2: matrix of difference
    D = ones(Nv,3);
    D(:,2) = (x1 - cx)'./h; % cx, cy, h computed before
    D(:,3) = (y1 - cy)'./h;
    c1 = (ones(1,Nv) - sum(D(:,2:3))*H)/Nv;
    GinvB = [c1; H];
    IminusP = eye(Nv)- D*GinvB;
    A2 = IminusP'*IminusP;
    % Record nonzero entries (i,j,s) for the sparse matrix
    [loci,locj,s] = find(A1 + A2);
    nnz = length(s);
    ii(curIdx:curIdx+nnz-1) = v(loci); % change to global index
    jj(curIdx:curIdx+nnz-1) = v(locj);
    ss(curIdx:curIdx+nnz-1) = s;
    curIdx = curIdx + nnz;
end
```

```
A = sparse(ii(1:curIdx-1),jj(1:curIdx-1),ss(1:curIdx-1),N,N);
```

In the subroutine `assemblematrix`, we use build in function `sparse` to form the sparse matrix A for the sake of memory. In this way, we avoid updating a sparse matrix inside a large loop. It is much faster than the code in [121]. Because the stiffness matrix in [121] is stored as a full matrix. It will be out of memory for a large number N, where N is the number of nodes. The code in lines $1-8$ is some initialization step. After line 9, we will get the vectorization form of the stiffness matrix for each element. The number of vertices for each element denotes Nv in the code. `circshift` is used for getting the neighbor for each node. Given `node` and `element`, it is easy to get the area, the diameter and centroid of each element. The centroid of a non-self-intersecting closed polygon defined by Nv vertices $(x_1, y_1), \dots, (x_{Nv}, y_{Nv})$ is the point (C_x, C_y), then

$$C_{\mathrm{x}} = \frac{1}{6E} \sum_{i=1}^{Nv} (x_i + x_{i+1})(x_i\, y_{i+1} - x_{i+1}\, y_i)$$
$$C_{\mathrm{y}} = \frac{1}{6E} \sum_{i=1}^{Nv} (y_i + y_{i+1})(x_i\, y_{i+1} - x_{i+1}\, y_i)$$

where E denotes the polygon's signed area and is calculated using the formula $E = \frac{1}{2}\sum_{i=1}^{Nv}(x_i\, y_{i+1} - x_{i+1}\, y_i)$ [15]. The diameter of each element h could be approximated by the square root of the area. From lines $10-24$, we get the area, diameter, centroid and normal direction for each element. The local stiffness matrix is given in lines $25-37$. For the first part in Equation (A.13), we can compute accurately. For each basis function φ_i, we define s_i^α as the coefficients of $\Pi^\nabla \varphi_i$ in the basis m_α:

$$\Pi^\nabla \varphi_i = \sum_{\alpha=1}^{3} s_i^\alpha m_\alpha, i = 1, \dots, 3. \tag{A.17}$$

where $m_1 = 1, m_2 = \frac{x-C_x}{h}, m_3 = \frac{y-C_y}{h}$. From equation (A.17) we have

$$(\nabla\Pi^\nabla\varphi_i, \nabla\Pi^\nabla\varphi_j)_{0,E} = \sum_{\alpha=1}^{3}\sum_{\beta=1}^{3} s_i^\alpha s_j^\beta (\nabla m_\alpha, \nabla m_\beta)_{0,E} = \sum_{\alpha=2}^{3}\sum_{\beta=2}^{3} s_i^\alpha s_j^\beta.$$

Note that $(\nabla m_1, \nabla m_1)_E = 0$ and $(\nabla m_\alpha, \nabla m_\beta)_E = 1, \alpha = 2, 3, \beta = 2, 3$, then the second equality holds. In addition, we can get

$$s_i^2 = (\nabla m_2, \nabla\varphi_i)_E = \int_{\partial E} \frac{\partial m_2}{\partial n}\varphi_i = \frac{1}{2h}(n^1(e_i) + n^1(e_{i+1}))$$

from Eq.(A.14) and similarly,

$$s_i^3 = (\nabla m_3, \nabla\varphi_i)_E = \int_{\partial E} \frac{\partial m_3}{\partial n}\varphi_i = \frac{1}{2h}(n^2(e_i) + n^2(e_{i+1})),$$

where $n = (n^1, n^2)$ is the normal direction for each edge and the intersection of two edges e_i and e_{i+1} is the vertex i.

Combing equation (A.15) and (A.17), we get $s^\alpha = (s_1^\alpha, \dots, s_{Nv}^\alpha), \quad \alpha = 1, \dots, 3$. Therefore we define a matrix H with size $2 \times Nv$ such that $H(1, :) = s^2$, $H(2, :) = s^3$, then $(\nabla\Pi^\nabla\phi_i, \nabla\Pi^\nabla\phi_j)_E$ could be written as $[H^T H]_{ij}$.

For the second part in Eq. (A.13), we know $\sum_{r=1}^{Nv} dof_r((I - \Pi^\nabla)\varphi_i) dof_r((I - \Pi^\nabla)\varphi_j) = [(I - \Pi^\nabla)^T (I - \Pi^\nabla)]_{ij}$, where $\Pi^\nabla = DG^{-1}B$,matrix D is $Nv \times 3$ and $G^{-1}B$ is $3 \times Nv$. It is straightforward to implement matrix D in code. The first column of D is 1, the second column of D is the value of m_2 in different vertices and the third column of D is the value of m_3 in different vertices. Finally we need to compute $G^{-1}B$. Based on deep understanding each component of the matrices formed from Eq. (A.16), we could rewrite $G^{-1}B = [s^1; s^2; s^3] = [s^1; H]$. The only thing is solving s^1. By the definition of P_0 and $P_0 v_h = \sum_{i=1}^{Nv} \mathrm{dof}_i(v_h)/Nv$,then we obtain $\sum(\Pi^\nabla\varphi_i)(V) = \sum\varphi_i(V)$, finally $\sum_{i=1}^{Nv}(s_i^1 + s_i^2 m_2(V_i) + s_i^3 m_3(V_i)) = 1$ from equation (A.17),that is, each component of s^1 could be $s_i^1 = (1 - (s_i^2 m_2(V_i) + s_i^3 m_3(V_i)))/Nv$, see line 34. Then a list of index and nonzero entries of local stiffness matrix are recorded in line $38 - 45$, which are used for updating a sparse matrix outsider the inner `for` loop. The global stiffness matrix is stored as sparse matrix in order to save the memory.

But there exists a large `for` loop in the subroutine `assemblematrix`. When the number of elements is large, it can quickly add significant overhead. We then use the vectorization technique to avoid the large `for` loop for the sake of efficiency. Since the length of each element is different, the vectorization could also be used with the same length of the elements. Here is the most efficient code for the assembling of stiffness matrix in two dimensions.

```
function A = assemblevem(node,elem)
%% Assemble the matrix equation
N = size(node,1);
elemVertexNumber = cellfun('length',elem);
nnz = sum(elemVertexNumber.^2);
ii = zeros(nnz,1); %initialization
jj = zeros(nnz,1);
ss = zeros(nnz,1);
index = 0;
for Nv = min(elemVertexNumber):max(elemVertexNumber)
   % find polygons with Nv vertices
   idx = find(elemVertexNumber == Nv); % index of elements
   NT = length(idx); % the # of elements
   % vertex index and coordinates
   vertex = cell2mat(elem(idx));
   x1 = reshape(node(vertex,1),NT,Nv);
   y1 = reshape(node(vertex,2),NT,Nv);
   x2 = circshift(x1,[0,-1]);
   y2 = circshift(y1,[0,-1]);
   % Compute geometry quantity: edge, normal, area, center
   bdIntegral = x1.*y2 - y1.*x2;
   area = sum(bdIntegral,2)/2; % the area per element
```

```
    h = repmat(sqrt(abs(area)),1,Nv); % h = sqrt(area)
    cx = sum((x1+x2).*bdIntegral,2)./(6*area); % the centroid
    cy = sum((y1+y2).*bdIntegral,2)./(6*area); % the centroid
    normVecx = y2 - y1; % normal vector
    normVecy = x1 - x2;
    % matrix B, D, I - P
    Bx = (normVecx + circshift(normVecx,[0,1]))./(2*h);
    By = (normVecy + circshift(normVecy,[0,1]))./(2*h);
    Dx = (x1 - repmat(cx,1,Nv))./h; % m(x) = (x - cx)/h
    Dy = (y1 - repmat(cy,1,Nv))./h;
    c1 = (1 - (repmat(sum(Dx,2),1,Nv).*Bx + repmat(sum(Dy,2),1,Nv).*By))/Nv;
    IminusP = zeros(NT,Nv,Nv);
    for i = 1:Nv
       for j = 1:Nv
          IminusP(:,i,j) = - c1(:,j) - Dx(:,i).*Bx(:,j) - Dy(:,i).*By(:,j);
       end
       IminusP(:,i,i) = ones(NT,1) + IminusP(:,i,i);
    end
    % assemble the matrix
    for i = 1:Nv
       for j = 1:Nv
          ii(index+1:index+NT) = vertex(:,i);
          jj(index+1:index+NT) = vertex(:,j);
          ss(index+1:index+NT) = Bx(:,i).*Bx(:,j) + By(:,i).*By(:,j) + dot(
              IminusP(:,:,i),IminusP(:,:,j),2);
          index = index + NT;
       end
    end
end
A = sparse(ii,jj,ss,N,N);
```

The out `for` loop goes over elements with the same the number of vertices,which is a big improvement compared with all the elements. The vectorization code is similar to `assemblematrix`. The major difference is vectorizing the matrix $I - DG^{-1}B$. We introduce three dimensional array to store, see lines $34 - 40$.

For the right hand side $\boldsymbol{b}$, we have

$$\boldsymbol{b} \approx \sum_{E \in \mathcal{T}_h} |E| f(C_x, C_y)/Nv.$$

Here we use the build in command `accumarray` in Matlab to avoid the `for` loop over all elements. The following codes are the whole assembling procedure:

```
function [u,A] = PoissonVEM(node,elem,pde)
%% Assemble the matrix equation
N = size(node,1);
elemVertexNumber = cellfun('length',elem);
nnz = sum(elemVertexNumber.^2);
ii = zeros(nnz,1); %initialization
jj = zeros(nnz,1);
ss = zeros(nnz,1);
b = zeros(N,1);
edge = zeros(sum(elemVertexNumber),2);
index = 0;
edgeIdx = 1;
for Nv = min(elemVertexNumber):max(elemVertexNumber)
   % find polygons with Nv vertices
   idx = find(elemVertexNumber == Nv); % index of elements
   NT = length(idx); % the # of elements
   % vertex index and coordinates
   vertex = cell2mat(elem(idx));
   x1 = reshape(node(vertex,1),NT,Nv);
   y1 = reshape(node(vertex,2),NT,Nv);
   x2 = circshift(x1,[0,-1]);
```

```
y2 = circshift(y1,[0,-1]);
% record edges
nextIdx = edgeIdx + NT*Nv;
newEdgeIdx = edgeIdx:nextIdx-1;
edge(newEdgeIdx,1) = vertex(:); % get edge per element
vertexShift = circshift(vertex,[0,-1]);
edge(newEdgeIdx,2) = vertexShift(:);
edgeIdx = nextIdx;
% Compute geometry quantity: edge, normal, area, center
bdIntegral = x1.*y2 - y1.*x2;
area = sum(bdIntegral,2)/2; % the area
h = repmat(sqrt(abs(area)),1,Nv); % h = sqrt(area)
cx = sum((x1+x2).*bdIntegral,2)./(6*area); % the centroid
cy = sum((y1+y2).*bdIntegral,2)./(6*area); % the centroid
normVecx = y2 - y1; % normal vector
normVecy = x1 - x2;
% matrix B, D, I - P
Bx = (normVecx + circshift(normVecx,[0,1]))./(2*h);
By = (normVecy + circshift(normVecy,[0,1]))./(2*h);
Dx = (x1 - repmat(cx,1,Nv))./h; % m(x) = (x - cx)/h
Dy = (y1 - repmat(cy,1,Nv))./h;
c1 = (1 - (repmat(sum(Dx,2),1,Nv).*Bx + repmat(sum(Dy,2),1,Nv).*By))/Nv;
IminusP = zeros(NT,Nv,Nv);
for i = 1:Nv
   for j = 1:Nv
      IminusP(:,i,j) = - c1(:,j) - Dx(:,i).*Bx(:,j) - Dy(:,i).*By(:,j);
   end
   IminusP(:,i,i) = ones(NT,1) + IminusP(:,i,i);
end
% assemble the matrix
for i = 1:Nv
   for j = 1:Nv
      ii(index+1:index+NT) = vertex(:,i);
```

```
         jj(index+1:index+NT) = vertex(:,j);
         ss(index+1:index+NT) = Bx(:,i).*Bx(:,j) + By(:,i).*By(:,j) + dot(
            IminusP(:,:,i),IminusP(:,:,j),2);
         index = index + NT;
      end
   end
   % compute the right hand side
   ft = area.*pde.f([cx cy])/Nv;
   b = b + accumarray(vertex(:),repmat(ft,Nv,1),[N,1]);
end
A = sparse(ii,jj,ss,N,N);
%% Find boundary edges and nodes
totalEdge = sort(edge(:,1:2),2);
[i,j,s] = find(sparse(totalEdge(:,2),totalEdge(:,1),1));
bdEdge = [j(s==1), i(s==1)]; % find the boundary edge
isBdNode = false(N,1);
isBdNode(bdEdge) = true;
bdNode = find(isBdNode); % get the boundary node
%% Impose boundary conditions
u = zeros(N,1);
u(bdNode) = pde.g_D(node(bdNode,:));
b = b - A*u;
%% Solve Au = b
freeNode = find(~isBdNode); % get the interior node
u(freeNode) = A(freeNode,freeNode)\ b(freeNode);
```

Note that we impose the Dirichlet boundary conditions, which is realized by lines $74-76$. Problems involving Neumann boundary part can also be implemented in a similar way. The boundary integral involving the Neumann boundary part is added into the right hand side. It is also vectorized using `accumarray`.

A.2.3 Numerical Results

In Figure A.1, it shows the irregular mesh from [122] in the rectangle domain $[0,1] \times [0,1]$. We shall define the following errors:

$$\|u_I - u_h\|_A = \sqrt{(u_I - u_h)^T \boldsymbol{A} (u_I - u_h)},$$
$$\|u_I - u_h\|_\infty = \max |u_I - u_h|,$$

where u_h is the numerical solution obtained by the linear virtual element methods; u_I is the nodal interpolation of the exact solution u in $\mathcal{T}_h$ and $\boldsymbol{A}$ is the corresponding stiffness matrix.

We choose the exact solution is $u = \cos(\pi x)\cos(\pi y) - 1$ and $f = 2\pi^2 \cos(\pi x)\cos(\pi y)$. The errors of $||u_I - u_h||_A$ and $||u_I - u_h||_\infty$ are presented in Table A.1. We could conclude that the order of accuracy in energy norm is near 1 and in maximum norm is almost 2. It verifies that virtual element method is feasible for solving Poisson equations with polygonal meshes in two dimensions. Finally, we also compare the corresponding assembling CPU time along with that in [121] in Table A.2, which reveals that our assembling time is not increasing too much for the large number of nodes. While it is more expensive to assemble the stiffness matrix due to the large `for` loop as the number of nodes is large enough in [121].

#Dofs	# of elements	$\|\|u_I - u_h\|\|_A$	$\|\|u_I - u_h\|\|_\infty$
34	16	8.2e-02	2.7e-02
130	64	5.9e-02	2.1e-02
514	256	2.4e-02	6.9e-03
2,044	1024	1.0e-02	1.6e-03
8,168	4096	4.9e-03	5.2e-04
32,652	20014	2.3e-03	1.2e-04
Rate		1.1	1.9

Table A.1: Errors measured in H^1 and L^∞ norms

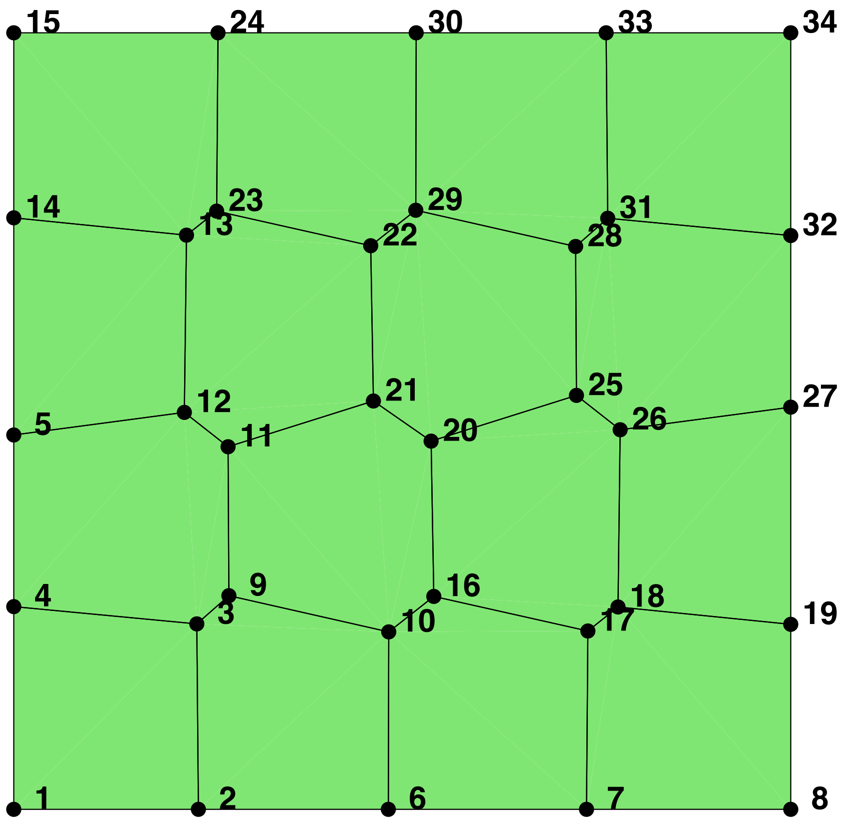


Figure A.1: The mesh when the number of the element is 16.

#Dofs	Assemble(our algorithm)	Assemble(other algorithm)
34	0.030971	0.0398962
130	0.018579	0.0416274
514	0.011572	0.194252
2,044	0.016255	0.521451
8,168	0.049402	3.4407
32,652	0.19699	40.2361

Table A.2: The comparison of assembling CUP time (in seconds)