A Robust Simplex Cut-Cell Method for Adaptive High-Order Discretizations of Aerodynamics and Multi-Physics Problems

by

Huafei Sun

B.A.Sc., University of Toronto (2007)

S.M., Massachusetts Institute of Technology (2009)

Submitted to the Department of Aeronautics and Astronautics in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2013

Author .... [illegible]
Department of Aeronautics and Astronautics
August 5, 2013

Certified by .... [illegible]
David L. Darmofal
Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .... [illegible]
Robert Haimes
Principal Research Engineer
Thesis Committee

Certified by .... [illegible]
Qiqi Wang
Assistant Professor of Aeronautics and Astronautics
Thesis Committee

Accepted by .... [illegible]
Eytan H. Modiano
Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

A Robust Simplex Cut-Cell Method for Adaptive High-Order Discretizations of Aerodynamics and Multi-Physics Problems

by

Huafei Sun

Submitted to the Department of Aeronautics and Astronautics
on August 5, 2013, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Despite the wide use of partial differential equation (PDE) solvers, lack of automation still hinders realizing their full potential in assisting engineering analysis and design. In particular, the process of establishing a suitable mesh for a given problem often requires heavy person-in-the-loop involvement. This thesis presents work toward the development of a robust PDE solution framework that provides a reliable output prediction in a fully-automated manner. The framework consists of: a simplex cut-cell technique which allows the mesh generation process to be independent of the geometry of interest; a discontinuous Galerkin (DG) discretization which permits an easy extension to high-order accuracy; and an anisotropic output-based adaptation which improves the discretization mesh for an accurate output prediction in a fully-automated manner.

Two issues are addressed that limit the automation and robustness of the existing simplex cut-cell technique in three dimensions. The first is the intersection ambiguity due to numerical precision. We introduce adaptive precision arithmetic that guarantees intersection correctness, and develop various techniques to improve the efficiency of using this arithmetic. The second is the poor quadrature quality for arbitrarily shaped elements. We propose a high-quality and efficient cut-cell quadrature rule that satisfies a quality measure we define, and demonstrate the improvement in nonlinear solver robustness using this quadrature rule. The robustness and automation of the solution framework is then demonstrated through a range of aerodynamics problems, including inviscid and laminar flows.

We develop a high-order DG method with a dual-consistent output evaluation for elliptic interface problems, and extend the simplex cut-cell technique for these problems, together with a metric-optimization adaptation algorithm to handle cut elements. This solution strategy is further extended for multi-physics problems, governed by different PDEs across the interfaces. Through numerical examples, including elliptic interface problems and a conjugate heat transfer problem, high-order accuracy is demonstrated on non-interface-conforming meshes constructed by the cut-cell technique, and mesh element size and shape on each material are automatically adjusted for an accurate output prediction.

Thesis Supervisor: David L. Darmofal
Title: Professor of Aeronautics and Astronautics

Acknowledgments

I would like to express my gratitude to the many people who have made this thesis possible. First, I would like to thank my advisor, Prof. David Darmofal, for giving me the opportunity to work with him and for his inspiration and encouragement throughout my graduate study. In addition, I would like to thank the other members of my thesis committee, Bob Haimes and Prof. Qiqi Wang. Special thanks go to Bob, for always being available to discuss all my questions and for always giving his very honest advice on everything. I would also like to recognize Michael Aftosmis and Prof. Krzysztof Fidkowski for providing critical feedback on the initial draft of this thesis and for sharing their experience on different topics. I am also very grateful to Dr. Steven Allmaras for his time devoted to ProjectX and for his help and insights on my work along the way.

This thesis would not have been possible without the efforts of the entire ProjectX team, past and present. I would like to thank them for their many contributions that enabled this work: Chris, Garret, and Todd, for laying the foundation of what ProjectX is today; Eric, JM, Laslo, and Masa, for their contributions to the ProjectX codes, their constant willingness to help, and their many constructive suggestions and insightful comments; Josh, for providing the bridge to the Boeing groups and for always being there to set up my case files and geometry files; Julie, for always being a teammate in good spirits about everything; Jun, Philip, and Steve O, for bringing fresh air to the team and for reminding me I've been here for too long. Special thanks go to Masa, who has always been available for discussion on any topic, research related or otherwise. I am indebted to him for all his help during my graduate study, especially on improving my talks, my papers, and this thesis. Thank you, and I wish you all the best for your academic career.

I would also like to thank everyone in ACDL for creating a productive yet fun environment to conduct research: Cuong, David M, Hemant, Tarek, and Xevi, for sharing their opinions and experience on a variety of topics covering both my research problems and their fields of expertise; Abby, for very kindly tolerating me as her officemate for one whole year; Xun, for being a close friend since my undergraduate life and for bringing a lot of fun in and outside the lab; Andrew, for organizing many lab events, especially those basketball games; Eric D, for always being responsive in fixing my computer issues; and all other ACDLers, in particular, Chad, Chelsea, David L, Han, Joel, Matt, Rui, and TC, for making my life in ACDL a lot richer. In addition, Jean, Robin, Sue, and Meghan deserve recognition for their help in organization and for quietly working behind the scenes.

I would also like to thank all my friends outside ACDL, which I apologize are too many to name here, for all the parties, game nights, lunches, dinners, and travels during these past few years. All of these made my life at MIT a lot more fun than it might otherwise have been. Special thanks are directed towards all my friends in Toronto, who always made my trips back to Toronto warm and memorable; you guys are terrific.

Last but not least, I would like to thank my family — Dad and Mom — for their perpetual and unconditional love and support. This has been a long and difficult road, and I would not have been able to reach the end of this road without their support and encouragement. I wish you all the happiness and health. In addition, I would like to send my most sincere thanks to my wife, Lang Yang. I don't know how to fully express my love and appreciation in words, but thank you for everything you have done for me over the years, and thank you for always standing by me especially when I was filled with doubts and frustrations. I love you, and look forward to continuing our journey together.

Finally, I would like to acknowledge the financial supports provided by the Boeing Company (technical monitor Dr. Mori Mani), NASA (NASA Award Number NNX12AJ75A, technical monitor Dr. Harold Atkins), the NSERC Postgraduate Scholarships Program, and MIT AeroAstro Department through fellowships.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Over the past several decades, numerical simulation has become an indispensable component in engineering analysis and design, and has been considered by many the third paradigm of scientific research, along with theory and experimentation. Being one important aspect of numerical simulation, partial differential equation (PDE) solvers have been widely used to analyze and study a variety of physical phenomena, ranging from fluid dynamics to solid mechanics to electromagnetics. Due to both algorithm development and increasing computational power, the complexity of problems – either from geometry, or physics, or both – that can be simulated has increased dramatically. However, despite the wide use of PDE solvers, lack of automation still hinders realizing their full potential. In particular, the process of establishing a suitable mesh for a given problem still often requires heavy person-in-the-loop involvement. This involvement is extensive in two stages of the process: making the decision of where a refined mesh resolution is needed, and generating a mesh that satisfies this decision and at the same time respects the geometry in question.

The process of manually specifying mesh resolution is not only costly in terms of person-hours, but also encounters great difficulties in leading to a reliable prediction of engineering outputs. In the context of computational fluid dynamics (CFD) simulations in the aerospace industry, the results from the AIAA Drag Prediction Workshops illustrate the difficulties in producing grid-converged results on even very fine meshes that are carefully "hand-crafted" by experts [84]. More specifically, Mavriplis [84] considered two families of meshes, and solved a transonic, turbulent flow over a wing using an industrial-strength CFD solver. The two mesh families were generated independently by NASA Langley and Cessna Aircraft Co., based on the best practices of each organization. A difference of seven drag counts was observed on the finest meshes of the two organizations, which had about ten times more degrees of freedom than typically used in practice. This difference is significant, as one drag count translates to four to eight passengers for a long-range passenger jet [48, 120]. Mavriplis concluded that fully resolving all features in computational aerodynamics problems is infeasible via successive global refinements of an arbitrary initial mesh. In other words, manually identifying all important features and specifying mesh resolution for an accurate output prediction can be very challenging, if possible at all, even on a geometry that is frequently encountered in practice.

Mesh adaptation provides a significant promise to reduce the amount of human intervention and also to produce a more reliable output prediction. However, mesh generation presents another barrier to an automated CFD simulation, especially when multiple meshes with different resolution requests are required in an adaptive process. In fact, mesh generation often represents the bottleneck in the CAD-to-mesh-to-solution cycle, as demonstrated for example in the context of applying CFD in industrial applications [40, 93]. For complex three-dimensional geometries in aerospace applications, generating a suitable mesh is particularly difficult, since highly anisotropic elements are often desired on the geometry surface for a sufficient boundary layer resolution. In addition, as higher-order discretizations receive more and more attention for CFD applications, elements on the geometry surface also have to be curved in order to maintain the benefit of higher-order discretizations [15].

Another difficulty for mesh generation comes from engineering applications that involve multiple materials separated by interfaces. These so-called interface problems are often governed by PDEs with discontinuous parameters across the material interfaces, or more generally, by different PDEs across the interfaces. Two examples in the aerospace industry include conjugate heat transfer (CHT) problems and fluid-structure interaction (FSI) problems. As the interface can be arbitrarily complex and curved or even moving, generating a suitable mesh is even more demanding than for a single-material problem. While the meshes on the two sides of the interface can require vastly different resolutions, they may be also required to match each other on the interface for a discretization method without special interface treatment.

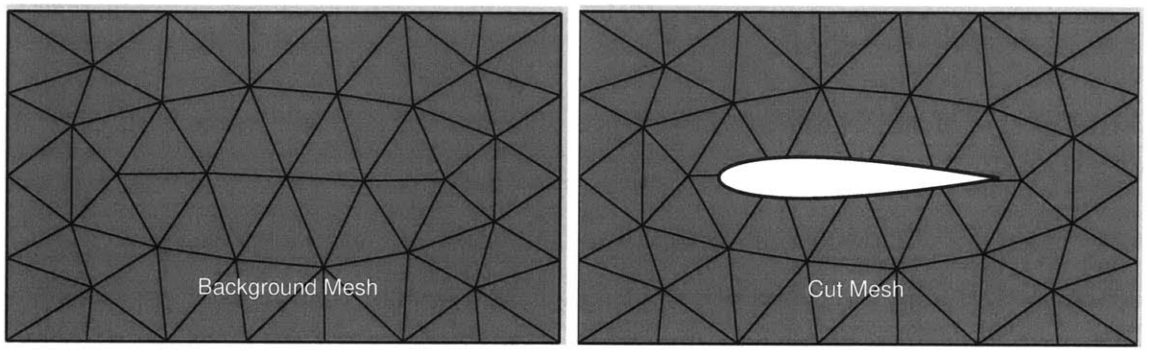
1.2 Thesis Objective

The objective of this thesis is to develop a robust PDE solution framework that provides a reliable output prediction in a fully-automated manner, and to assess the framework through a wide range of applications, including aerodynamics and multi-physics problems. To be fully-automated, the framework must not require user interactions or detailed previous knowledge at both mesh generation stage and solution stage; and to be robust, the framework must be able to provide reliable solutions for a wide range of geometries and physical conditions. The proposed solution framework incorporates a simplex cut-cell technique, a high-order discretization, and an anisotropic output-based adaptation. The motivation and background for each of these components are presented in the following section.

1.3 Background

1.3.1 Simplex Cut-Cell Method

The difficulties for the mesh generation mechanics, including those imposed by high anisotropy and curved geometries, motivate an alternative for a more automated and robust meshing process: cut cells. For a cut-cell mesh, the mesh generation process does not need to respect the geometry of interest. More specifically, a background mesh is first generated on a box without conforming to the geometry, which is often referred to as the embedded geometry. This background mesh then intersects with the embedded geometry, and the part that is outside of the computational domain is discarded, resulting in a cut mesh that has arbitrarily shaped elements inside the computational domain. An example is shown in Figure 1-1. This technique effectively simplifies the problem of meshing an arbitrary geometry to meshing a box, and hence significantly improves the robustness and automation for mesh generation. However, the discretization method now needs to account for the arbitrarily shaped elements on the cut-cell mesh.



(a) Background mesh (b) Cut mesh, with embedded geometry

Figure 1-1: Example of cut-cell mesh

Purvis and Burkhalter were the first to consider the cut-cell method [102], where they worked with the full potential equations on Cartesian background meshes. The Cartesian cut-cell method was later used extensively in the CFD community for treating problems with complex boundaries; see for example [2, 69, 131]. One prominent example is Cart3D [2], which is a three-dimensional solver for Euler equations and has been shown capable of handling very complex geometries [90]. However, the usage of Cartesian grids limits the achievable directions of element anisotropy, making the discretization of arbitrarily oriented features inefficient.

Recently, a simplex cut-cell method was developed by Fidkowski and Darmofal for embedded boundary problems in two and three dimensions [50]. They further combined the method with a high-order discontinuous Galerkin discretization, and represented the embedded geometries using cubic splines and quadratic patches in two and three dimensions, respectively. However, the original algorithm suffered from several weaknesses that limit the robustness and automation promised by the cut-cell method:

1. intersection ambiguity due to limitations of numerical precision;

2. poor quadrature quality for cut cells of arbitrary shapes, resulting in poor convergence behavior for nonlinear solvers;

3. small volume ratios between neighbor elements, causing inaccuracy and poor conditioning.

Items 2 and 3 were investigated in two dimensions by Modisette [87]. In particular, Item 2 was tackled by recognizing "canonical" shapes for three- and four-side cut elements. However, extension of this approach to three dimensions is not trivial, as will be discussed in more detail in Chapter 3. Item 3 was greatly mitigated by merging the two neighbor elements together, and this approach is equally applicable in three dimensions. In this thesis, we will demonstrate the limitations due to Items 1 and 2 in three dimensions, and propose solutions for these items to improve the overall robustness of the method.

For interface problems, removing the constraint of being interface-conforming would also greatly improve the robustness and automation of the meshing process. In fact, many methods have been proposed on non-interface-conforming meshes, for example, the immersed interface method [73], the ghost fluid method [47], and the immersed finite element method [74]. A more complete literature review will be provided in Chapter 6. However, these methods remain largely second-order accurate, and/or extension to multi-physics problems with a more general interface coupling remains non-trivial. As the cut-cell technique has shown success for problems with complex boundaries, we will extend this technique to solve multi-material and multi-physics problems in this thesis.

1.3.2 Adaptive High-Order Discretization

High-order methods are characterized by their ability to achieve higher fidelity at a lower cost. In general, a discretization method has its error converging as $E \sim \mathcal{O}(h^r)$, where E is some measure of error, e.g. L^2-error, and h is some measure of mesh size. High-order methods aim to improve the simulation accuracy by increasing the convergence rate r, and in this work, high-order methods are those that achieve $r > 2$ (for L^2-error). While these methods are known to provide a high convergence rate for smooth problems, for problems with singularities, the convergence rate r is often limited by solution regularity. In order to realize the benefit of high-order methods for these problems, the effect of singularities has to be controlled, by for example adaptive meshes. A well-know example is the Poisson problem on an $L-$shaped domain, which can be treated on graded meshes for better convergence [29]. For more complex problems with singularities, Yano *et al.* achieved high-order convergence by combining a high-order discretization with adaptation in the aerodynamics context [128].

The finite element framework provides a conceptually easy path to high-order methods by increasing the degree of basis polynomials. In this work, a discontinuous Galerkin (DG) method is employed. As the DG method imposes element coupling and boundary conditions only through face fluxes, it easily permits elements of arbitrary shapes, which are needed for using non-geometry-conforming meshes. Further, the DG method allows solution discontinuity across elements, and thus has an element-wise compact support of basis functions.

The first DG method was introduced by Reed and Hill for scalar hyperbolic equations [103], and its error analysis was later provided by Johnson and Pitkäranta [65] and Richter [104]. The method was extended for nonlinear hyperbolic problems by incorporating Godunov's flux [28]. Cockburn and Shu and their co-authors combined the DG spatial discretization with a Runge-Kutta explicit time integration for non-linear systems of hyperbolic equations, and presented their method in a series of papers [30, 32, 33, 35]. Separately, Allmaras and Giles presented a second-order DG method for Euler equations [4, 5]. A review of the early development of DG methods is provided by Cockburn *et al.* [31].

Independent of the development for hyperbolic problems, DG was also developed for elliptic problems, beginning with the interior penalty methods [6, 9, 122]. More recently, Bassi and Rebay developed a DG discretization of diffusive operator, known as BR1 [16]. They later improved BR1 and presented the so-called BR2 method [17], which achieves stability for purely elliptic problems and recovers an element-wise compact stencil. Cockburn and Shu generalized the BR1 method and introduced the local discontinuous Galerkin (LDG) method [34]. It was then modified by Peraire and Persson to recover an element-wise compact stencil, yielding the compact discontinuous Galerkin (CDG) method [96]. A unified analysis of DG methods for elliptic problems is provided by Arnold *et al.* [7].

1.3.3 Output-Based Error Estimation and Adaptation

Engineering applications often require an accurate prediction of certain output quantities. Output-based adaptation provides an autonomous means to reduce the error in an output to a specified level. The adaptive framework is illustrated in Figure 1-2, where as inputs, a governing PDE, an output of interest, an error tolerance, and a maximum allowable run-time are specified. From these inputs, the PDE is discretized on an initial (typically coarse) grid, and the error in the output solution is estimated. If the error and time tolerances are not met, the output error is then localized to elements, and an adapted mesh is generated according to the localized error, e.g. by refining the elements causing large errors. The whole process then repeats until either the error tolerance is met or time is exhausted. The two key components in the adaptive framework are the output error estimation and the mechanics for improving the mesh quality to reduce the output error.

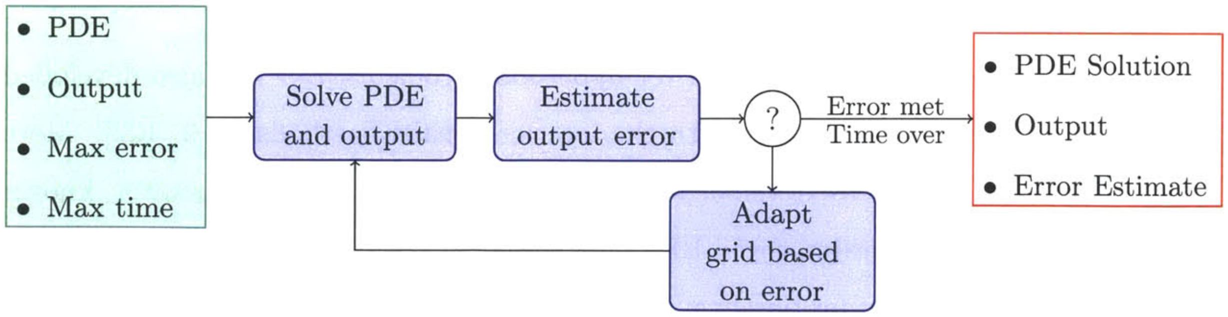


Figure 1-2: Illustration of the autonomous output-based adaptive framework

Error Estimation

In the adaptive framework, error estimation serves two critical functions: (1) estimating the global error to assess the quality of the discretized solution, and (2) localizing the error to elements and identifying the elements with large errors. *A posteriori* error estimation techniques have been developed over the past decades for these purposes. For instance, error estimates based on energy [3] and interpolation error [41] have been used for adaptation. However, these methods generally fail for hyperbolic problems, where upstream errors can propagate downstream [116].

More recently, output-based error estimation techniques have been developed. These techniques estimate and localize the output error by explicitly incorporating the dual problem associated with the output. The solution to the dual problem, the adjoint, links local residuals to the output error, and hence identifies the regions that are important for the accurate prediction of the output. In this work, the dual-weighted residual (DWR) method proposed by Becker and Rannacher [18, 19] is used. The method has been applied to a wide range of engineering applications; see for example [10, 53].

Anisotropic Output-Based Adaptation

Given an error indicator, the goal of adaptation is to decrease the error by modifying the discretization mesh. The DWR method provides a localized output error for each element, which is sufficient for an isotropic output-based adaptation. For example, a fixed-fraction strategy can be employed, where a fixed percentage of elements that have the largest error are refined and those with the smallest error are coarsened. However, the localized error alone does not provide enough information for an anisotropic adaptation. Anisotropic information is often formulated as metric tensor field, which contains the information of size and orientation of each element; see for example [57, 121]. This field is taken as input by many anisotropic mesh generators, for example, in BAMG [21, 59] and EPIC [85]. Therefore, an anisotropy detection strategy that provides local metric tensors is also required in the adaptation framework.

Motivated by the fact that interpolation errors are closely related to solution Hessian for linear interpolations, Peraire *et al.* [95] introduced an anisotropy detection method based on estimating the Hessian of a scalar solution field. Venditti and Darmofal [121] combined this technique (using the Hessian of the Mach number) with the DWR method, and proposed an anisotropic output-based adaptation algorithm for the compressible Navier-Stokes equations. Fidkowski and Darmofal [50] later generalized the idea to high-order discretizations based on high-order derivatives of the Mach number. A variant of the strategy based on the jump in Mach number across elements was proposed for quadrilateral elements with a DG discretization by Leicht and Hartmann [72]. While these methods have been successful for anisotropic adaptation, the choice of Mach number is arbitrary, and more importantly, the anisotropy of the adjoint is not taken into account.

Recently, Yano and Darmofal proposed the Mesh Optimization via Error Sampling and Synthesis (MOESS) algorithm for anisotropic output-based adaptation [124, 127]. The algorithm casts the adaptation problem as a continuous constrained optimization problem with the design variable being the metric tensor field $\mathcal{M}$ and the objective function being the output error $\mathcal{E}$. The sensitivity, $\mathcal{E}'[\mathcal{M}](\partial\mathcal{M})$, is approximated through local error re-computation on different configurations due to edge splits for each element. This strategy resolves the aforementioned shortcomings of the Hessian-based methods, as it incorporates both primal and dual solution behaviors by directly monitoring the output error, and treats low regularity features more robustly by removing any *a priori* assumption on error convergence behavior. While this method has been shown superior to the Hessian-based methods, it is not readily suitable for elements of arbitrary shapes. This is because for those elements, the duality between the metric and the mesh as well as the error sampling through local solves is not well defined. In this thesis, we will extend this method to handle cut-cell meshes for both embedded boundary and interface problems.

For multi-physics problems, which involve multiple sub-domains governed by different PDEs, an adaptive scheme needs to consider the entire coupled system and employ appropriate mesh resolution on each sub-domain. For these problems, several researchers have recently combined the DWR output error estimation with local adaptive mesh refinement schemes [71, 105], and have demonstrated the framework on (Cartesian) interface-conforming meshes for simple geometries. In this thesis, we will demonstrate our anisotropic output-based adaptation scheme on simplex non-interface-conforming meshes through a conjugate heat transfer problem.

1.4 Thesis Overview

This thesis presents work toward the development of a robust PDE solution framework that provides a reliable output prediction in a fully-automated manner. In particular, the framework consists of a simplex cut-cell technique, a high-order DG discretization, and an output-based adaptation. The primary contributions of the thesis are as follows.

- Development of a robust intersection algorithm for cut cells in three dimensions, with an efficient use of adaptive precision arithmetic.
- Development of a high-quality and efficient quadrature rule for arbitrary shapes in two and three dimensions, and demonstration of the robustness improvement in nonlinear solvers using the proposed quadrature rule.

- Demonstration of the robustness and automation of the framework for a range of three-dimensional aerodynamics problems, including inviscid and laminar flows.

- Extension of the MOESS adaptation algorithm to cut-cell meshes for embedded boundary and interface problems.

- Development of a high-order discontinuous Galerkin method with a dual-consistent output evaluation for elliptic interface problems, and demonstration of high-order accuracy on non-interface-conforming meshes constructed by the cut-cell technique.

- Extension of the elliptic interface strategy (as a monolithic approach) to multi-physics problems, and demonstration through a conjugate heat transfer problem, where output-based adaptation adjusts mesh element size and shape on each material in a fully-automated manner.

The thesis is organized as follows. Chapter 2 presents the efficient use of adaptive precision arithmetic for robust cut-cell intersections. Chapter 3 proposes a high-quality quadrature rule for cut cells. Chapter 4 first provides the background of the DG discretization and the DWR output error estimation, and then extends the MOESS algorithm to handle cut-cell meshes. In Chapter 5, we first show the impact of an improved quadrature quality on the convergence behavior of the nonlinear solver, and then demonstrate the robustness and automation of our framework through a range of three-dimensional aerodynamics problems. Chapter 6 derives the DG method for elliptic interface problems, and extends the method to non-interface-conforming meshes using the cut-cell technique. Chapter 7 presents the DG method for multi-physics problems, and demonstrates the developed solution framework through a conjugate heat transfer problem.

Chapter 2

Robust Intersection for Cut-Cell Mesh Generation

While the cut-cell method simplifies the problem of meshing an arbitrary geometry to meshing a box, a robust intersection algorithm is a fundamental requirement for the method to be fully automated. For any input of a geometry definition and a background mesh that does not conform to the geometry, the algorithm has to always yield a topologically consistent cut mesh that defines a valid tessellation of the computational domain. In Section 2.1, we briefly review the robustness issues in computational geometry in general, and introduce adaptive precision arithmetic to overcome these issues. Section 2.2 gives a brief overview of the intersection algorithm. In Section 2.3, we show that an implementation with epsilon-tweaking is not sufficient for a robust cut-cell construction, and demonstrate that using the adaptive precision arithmetic does ensure intersection correctness but is unacceptably slow. Section 2.4 describes the many techniques developed to make this arithmetic computationally affordable for cut-cell intersections. Results are shown in Section 2.5 to demonstrate the robustness and efficiency of the developed intersection algorithm. A detailed description of the entire algorithm including pseudocodes is provided in Appendix B, with an emphasis on how the adaptive precision arithmetic is used in every step of the intersection. Note in this chapter, we assume the geometry definition represents an embedded domain boundary. If the geometry represents an embedded interface instead, the developed intersection algorithm is still applicable, but the cut mesh needs to be constructed on both sides of the geometry.

2.1 Background

In computational geometry, most algorithms are designed and proven to be correct in the context of assuming all arithmetic on real numbers is exact. When the exact arithmetic is replaced by finite-precision arithmetic in implementation, for example the IEEE standard 754 floating-point arithmetic, many geometric algorithms can lead to unpredictable failures, including inconsistent or self-contradictory geometry structures. A geometric algorithm or implementation with such failures for certain inputs is often called non-robust. In the context of cut-cell construction, a non-robust intersection algorithm does not always return a consistent cut mesh, and thus would eliminate the potential of full automation promised by the proposed PDE solution strategy.

The difficulties in making geometric algorithms robust come from the fact that in addition to numerical outputs, geometric algorithms also need to return combinatorial structures that are consistent with the numerical outputs. Citing a notation used by Yap [129], geometric computing can be decomposed into two parts:

$$\text{Geometric Computing} = \text{Combinatorial} + \text{Numerical Computing}.$$

The combinatorial part derives geometric relationship among geometric entities, for example, it answers the question whether a point lies inside a given triangle. The numerical part finds the numerical values of geometric entities, such as coordinates of intersection points. One key step in the combinatorial part is to evaluate geometric predicates, which are conditional tests that branch the algorithm into different topologies (or geometry structures), and only one of these topologies corresponds to the theoretical result. The conditional tests almost always involve numerical calculations, and if all the calculations for every predicate are evaluated using exact arithmetic, the algorithm will always be in a state equivalent to its theoretical counterpart. The robustness of a geometric algorithm thus lies in **every** predicate being answered correctly.

In an implementation, the predicate often involves comparing two numerical values. Without loss of generality, we can assume one of the values is zero, i.e. the predicate needs to query the sign of certain expression based on some numerical input $f(\langle input \rangle)$, and hence the key ingredient in robust geometric algorithms is a correct sign computation. While we query the sign of f, we have only its approximation $\tilde{f}$ due to finite precision, where $|\tilde{f} - f| < \delta$ and δ is defined by the precision involved. Evaluating the predicate can therefore only be based on the approximation $\tilde{f}$. Unlike numerical computation where the round-off errors in approximating f can be tracked and quantified, the effect of round-off errors is often unpredictable for geometric computation due to the combinatorial nature of predicates. A widely used method to decide $f > 0$ given $\tilde{f}$, often named epsilon-tweaking, is based on an introduced ϵ_{magic}:

$$f > 0 \Longleftrightarrow \tilde{f} > \epsilon_{\text{magic}},$$

where ϵ_{magic} is usually chosen by trial and error, i.e. it is adjusted until no catastrophic output happens for the tested inputs. However, choosing an ϵ_{magic} that works for any input is tedious and non-trivial, if possible at all. As one incorrect sign computation may result in a wrong topology, using epsilon-tweaking for many geometry problems can lead to severe robustness issues. This will be demonstrated in Section 2.3 for the cut-cell intersection problem.

For a couple of decades, significant research effort has been devoted to tackle the robustness and reliability issues in computational geometry. Schirra [110] and Hoffmann [61] provide excellent overviews about the development in this field. One way is to design geometric algorithms that can deal with imprecise inputs and calculations. A simple example is to represent every straight line by a tubular region with thickness defined based on round-off errors. The result of this approach is not always exact, i.e. same as the theoretical result, but consistency between the combinatorial structures and numerical outputs is ensured. While this approach has shown success for a small number of problems, for example intersection of polygons as demonstrated by Milenkovic [86], there is no general theory on how to design geometric algorithms with imprecision. Guibas provides a summary of current difficulties in pursuing this route [55].

Another route toward robust geometric algorithms is through exact geometric computation (EGC), where geometric predicates are always correctly evaluated. An obvious EGC approach is to compute every number involved in the predicates using exact arithmetic. Note, however, that this does not require exact representations for all numerical outputs in the algorithm. As numerical inputs are almost always rationals given as floating-point numbers (or integers), an exact rational arithmetic can eliminate the robustness issue when operations involve only $+$, $-$, $\times$, and $\div$. However, such an arithmetic is 10,000 times slower than floating-point arithmetic for a Delaunay triangulation as reported in [68].

While computing every expression f for predicates using exact arithmetic is slow, it is recognized that using its approximation $\tilde{f}$ instead of f is sufficient for sign computation for most of cases. Failure occurs only in certain degenerate or nearly-degenerate cases. Inspired by this fact, adaptive precision arithmetic has been developed, where precision is refined to be just sufficient for the theoretical correctness of the geometric algorithm. Shewchuk developed such an arithmetic using a multi-component format for storing floating-point numbers [112], but his arithmetic supports only $+$, $-$, and $\times$, and our cut-cell intersection problem requires polynomial root-finding as shown in Section 2.3. Yap and Dubé [44] and Burnikel *et al.* [26] encoded adaptive precision arithmetic in the libraries CORE and LEDA, respectively. Both execute exact sign computation with the help of separation bounds for algebraic numbers (roots of polynomials with rational coefficients), and hence support root-finding for such polynomials. Appendix A.1 briefly introduces the representation of algebraic numbers and their sign computation. In this work, the data type LEDA *real* is used for intersection. Although the correctness of sign computation is guaranteed using *real*, simply replacing standard *double* precision type by *real* is not computationally affordable as demonstrated in Section 2.3.

2.2 Intersection Overview

For three-dimensional applications in this work, curved surfaces are approximated by patches of quadratic triangles, which are proposed for cut-cell applications in Fidkowski's work [49]. Each quadratic patch is defined based on a parametric mapping from a unit reference triangle, via

$$\mathbf{x} = \sum_{i=1}^{6} \phi_i(\mathbf{X})\mathbf{x}_i, \tag{2.1}$$

where $\mathbf{X} \in \mathbb{R}^2$ is the coordinate in reference space, $\mathbf{x}_i \in \mathbb{R}^3$ are the coordinates of the six patch nodes (corner nodes plus edge midpoints) in physical space, and ϕ_i are the quadratic Lagrange polynomials defined on the reference triangle. Quadratic-patch representation guarantees a watertight geometry, and enables an analytical solution for the intersection problem. Geometry slope discontinuities are present between patches, but they can be controlled by refinement of the patches.

Starting with a quadratic-patch representation of the geometry and a simplex background mesh, the intersection algorithm constructs the topology of the cut mesh. An illustration is shown in Figure 2-1(a), where a background element intersects a quadratic-patch surface. Figure 2-1(b) shows the wire-frame of the two resulting cut elements, one of which is outside the computational domain and so not constructed by the intersection algorithm. The skeleton of the algorithm is similar to Fidkowski's implementation [49], which consists of four steps:

1. computation of intersection points, named *zerod* objects;

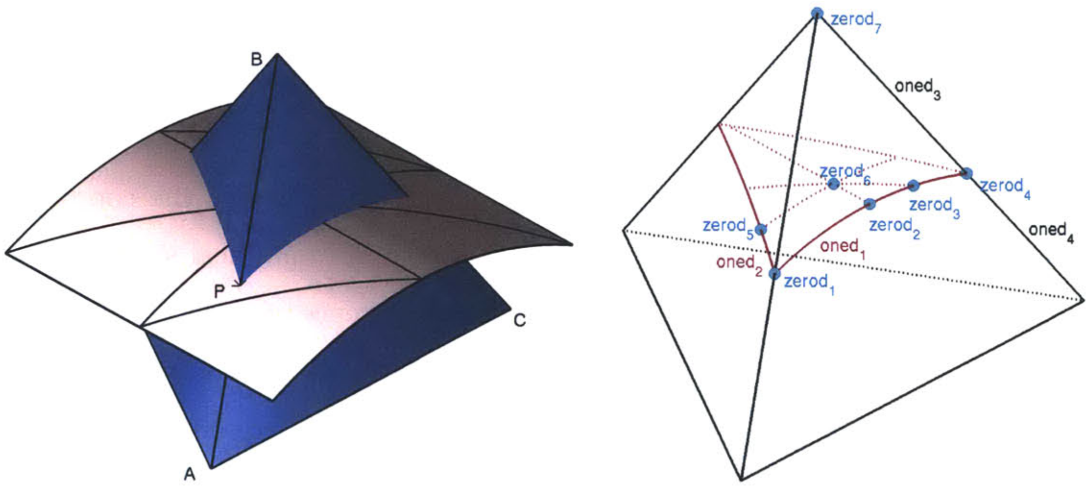
2. construction of intersection edges (*oned* objects) by ordering and connecting the

zerod objects;

3. construction of intersection faces (*twod* objects) by connecting the *oned* objects into loops;

4. construction of cut elements (*threed* objects) by making the *twod* objects into closed volumes.

Details of each step are provided in Appendix B, with an emphasis on our changes to Fidkowski's implementation. Most of the changes are for numerical conditioning concerns and for efficiency improvement when using adaptive precision arithmetic, which will be introduced and discussed in the rest of this chapter.



(a) (b)

Figure 2-1: Example intersection between a background tetrahedron and a quadratic-patch surface

2.3 Epsilon-Tweaking and Adaptive Precision Arithmetic

Because each step in the intersection algorithm is built upon the previous steps, a correct construction of the *zerod* objects lays the foundation for the whole algorithm.

One major component in the *zerod* object construction is to find the intersection points between tetrahedron edges and patches, for example, the intersection point P between the edge AB and the patches in Figure 2-1(a). Let the edge be represented by $\mathbf{x} = \mathbf{x}_A + t(\mathbf{x}_B - \mathbf{x}_A)$, and one quadratic patch is defined as in (2.1). Then the problem is: find X, Y, and t such that

$$\sum_{i=1}^{6} \phi_i(X,Y)\mathbf{x}_i = \mathbf{x}_A + t(\mathbf{x}_B - \mathbf{x}_A) \tag{2.2}$$

$$t \in [0,1], \quad X, Y \geq 0, \quad X + Y \leq 1, \tag{2.3}$$

which is a system of three quadratic equations in three variables and has an analytical solution. The question whether (2.2) has a solution in the range of (2.3) represents a geometric predicate that branches the algorithm into two different topologies. Only one of them is valid, and hence answering this predicate correctly is critical. A simple implementation is to solve (2.2) using an analytical formula (root-finding for cubic equations involved) and verify the constraints (2.3) for the found roots with epsilon-tweaking. Such an implementation is similar to that in the work of Fidkowski [49]. However, this method suffers from severe robustness issues. As an illustration, the intersection algorithm is applied to a patch representation of ONERA M6 wing with 6500 quadratic patches as in Figure 2-2(a). The intersection is carried out for 15 similar background meshes, each of which has about 7000 tetrahedra and results in about 1200 cut elements. Figure 2-2(b) shows one of the meshes. With the epsilon-tweaking method, eight of the meshes cannot return a topologically valid cut mesh due to incorrect evaluation of predicates, for example, the verification of the range (2.3).

The same algorithm is then implemented using LEDA *real*. One difficulty in using *real* is the need to deal with transcendental functions, which are not supported for algebraic numbers. The intersection curve between a plane and a quadratic patch is a conic in the reference space of the patch, as proven in [49]. As an illustration, the shaded *twod* object in Figure 2-3(a) is shown in the reference space of the parent patch in Figure 2-3(b). Both *oned*$_1$ and *oned*$_2$ result from an intersection between a plane

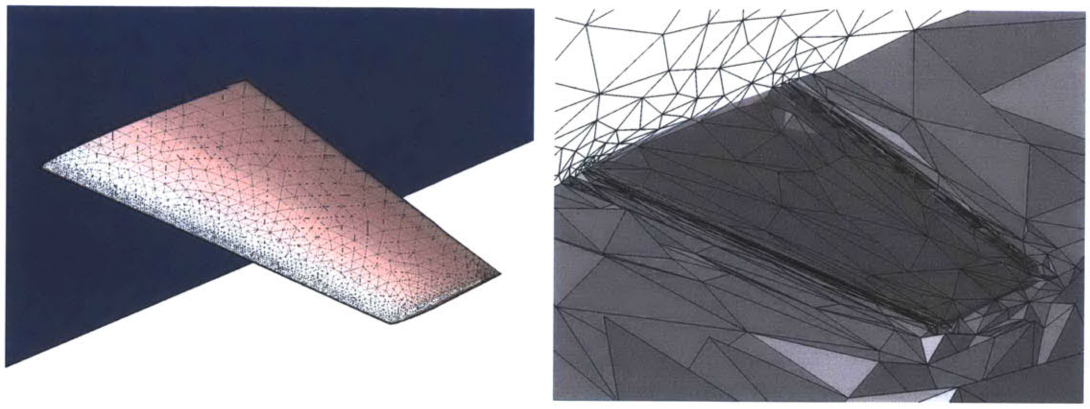
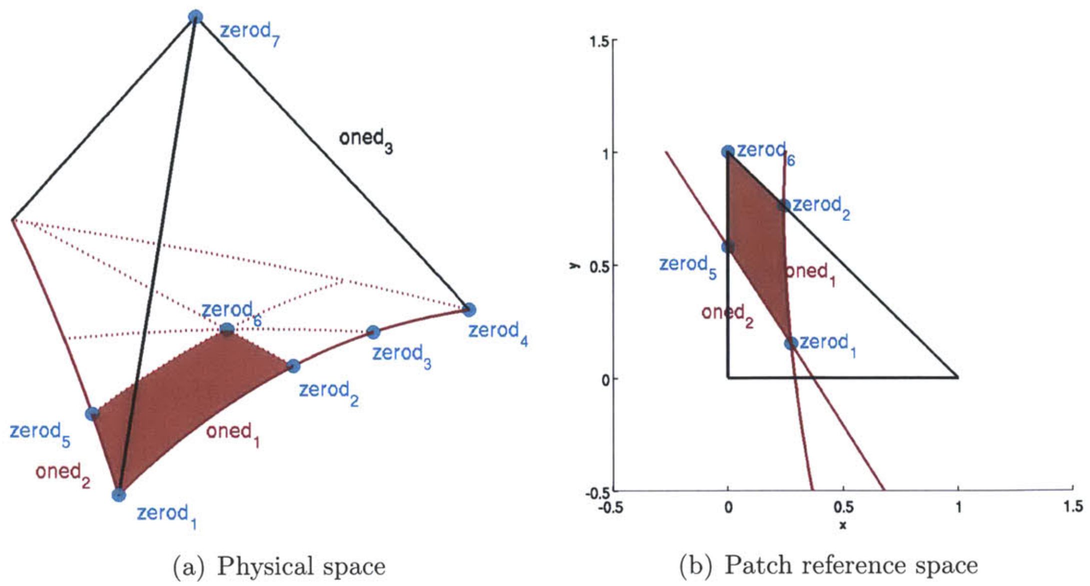

(a) (b)

Figure 2-2: Quadratic-patch representation of ONERA M6 wing and an example background mesh

and a patch, and are conic sections in Figure 2-3(b). For each conic section, all the *zerod* objects on it need to be ordered, and this needs a parameterization of the conic, which often requires transcendental functions as in Fidkowski's implementation [49]. One fix is to use a rational Bézier representation for conics [101], but a much simpler and more efficient method relies on the convex properties of conics. Johnston proved that the order of points on a convex segment is the same as the order on the convex hull formed by these points [66]. This effectively turns the ordering problem into ordering on a convex polygon, which needs only orientation tests and enables the use of LEDA *real*. Details are provided in the description of *oned* object construction in Appendix B.2.

With transcendental functions removed, all the *double* numbers are changed to LEDA *real*, and the entire intersection code is made "epsilon-free" by removing all the ϵ_{magic}'s. The same tests on ONERA M6 wing are carried out, and each of the 15 background meshes returns a valid cut mesh. However, constructing each one of these cut meshes takes even longer than the solution time of an inviscid transonic flow on the wing using the ProjectX DG solver [13, 43, 51].



(a) Physical space (b) Patch reference space

Figure 2-3: Example intersection in patch reference space

2.4 Efficiency Improvement

In the library of LEDA (or CORE), every algebraic number has its entire construction history stored as a directed acyclic graph (DAG), whose internal nodes represent arithmetic operations (e.g. $+$) and whose leaf nodes are the input numbers. Each internal node is also stored with its round-off error, and a positive number known as separation bound, which is used to ensure correct sign computation. More details (including an example of DAG) are provided in Appendix A.1, and separation bounds derived for LEDA *real* are described in [25]. When the precision of the number needs to be refined for sign computation, the whole DAG needs to be updated. Also, when the number in the query is exactly zero, the precision required is much higher, especially if root-finding and division are in the DAG as these operations have much looser separation bounds. Therefore, for efficiency concerns, *double* precision arithmetic should be used whenever possible, and if adaptive precision arithmetic has to be used, it is important to:

(1) keep the construction history of every number simple;

(2) avoid polynomial root-finding (if possible);

(3) keep the degree of every algebraic number low;

(4) avoid asking for the sign of a number that is exactly zero (if possible).

Various techniques are developed to make the adaptive precision arithmetic affordable for our cut-cell intersection problem. The key concepts are described in the rest of this section, and more discussion is given in Appendix A.2.

Intersection Detection

The most computationally expensive step in the intersection algorithm is in solving (2.2), which governs the intersection between each pair of a background tetrahedron edge and a patch. As most of these pairs do not intersect, it is appealing to quickly identify such a case without attempting to solve (2.2). Two tests are developed for this purpose.

The first is a bounding-box test, where an axis-aligned bounding box (AABB) is computed for each quadratic patch. Note the AABB is defined based on the extrema of the patch in each coordinate direction. Possible overlap between the AABB and a background tetrahedron edge (or face) is examined before attempting to solve the intersection problem. The method of separating axis, see for example [45], is applied for this examination. This method is a simple and efficient way for determining whether two convex sets intersect by projecting the two sets onto one or more lines. Further, because AABB's are not tight even when computed exactly, round-off errors are tolerable in this case. Thus, the bounding-box test is implemented using *double* precision.

The second test is based on polynomial root-bounding. Because the system (2.2) is linear in t, we can eliminate t easily and obtain a bivariate quadratic system in X

and Y. Let each equation in the system (2.2) be denoted by

$$\mathbf{F}^d \equiv \sum_{i=1}^{6} \phi_i(X,Y)\mathbf{x}_i^d - (\mathbf{x}_A^d + \alpha^d t) = 0, \tag{2.4}$$

where $\alpha^d \equiv \mathbf{x}_B^d - \mathbf{x}_A^d$, and $d \in \{1,2,3\}$ represents the dimension index of coordinates. Let $\tilde{d}$ denote the dimension for the largest α^d, i.e. $\tilde{d} \equiv \arg\max_d \alpha^d$, then a biquadratic system that has the same roots in X and Y as (2.4) can be obtained:

$$\begin{cases} S_1(X,Y) \equiv \alpha^{\tilde{d}}\mathbf{F}^{\mathrm{mod}(\tilde{d}+1,3)} - \alpha^{\mathrm{mod}(\tilde{d}+1,3)}\mathbf{F}^{\tilde{d}} = 0 \\ S_2(X,Y) \equiv \alpha^{\tilde{d}}\mathbf{F}^{\mathrm{mod}(\tilde{d}+2,3)} - \alpha^{\mathrm{mod}(\tilde{d}+2,3)}\mathbf{F}^{\tilde{d}} = 0 \end{cases}. \tag{2.5}$$

Note a mathematically equivalent formulation to Eq. (2.5) is

$$\begin{cases} \mathbf{F}^{\mathrm{mod}(\tilde{d}+1,3)} - \frac{\alpha^{\mathrm{mod}(\tilde{d}+1,3)}}{\alpha^{\tilde{d}}}\mathbf{F}^{\tilde{d}} = 0 \\ \mathbf{F}^{\mathrm{mod}(\tilde{d}+2,3)} - \frac{\alpha^{\mathrm{mod}(\tilde{d}+2,3)}}{\alpha^{\tilde{d}}}\mathbf{F}^{\tilde{d}} = 0 \end{cases}.$$

With adaptive precision arithmetic, this formulation can be prohibitively expensive for certain degenerate cases due to the division operator, for which the separation bound is less tight than for the other basic arithmetic operators [25].

As solving the system (2.5) requires root-finding for a polynomial that is at least cubic, detecting possible roots without solving leads to a significant reduction in computational cost. This is done by first computing the two Sylvester resultants of the bivariate system, each of which is a univariate quartic polynomial and has the same roots as the original system. This quartic polynomial is first classified based on its number of real roots, and then Sturm's Theorem is applied to the resultants to detect the existence of roots in $[0,1]$. This whole process does not involve any root-finding of polynomials. Further, the Sturm's sequence for a quartic polynomial can be expressed in terms of its discriminant and invariants, which are already computed for its classification; see Appendix A.3. Description of Sturm's Theorem and definitions of polynomial resultants can be found in algebra textbooks, for example, [130].

Conic-Conic Intersection

The obtained bivariate quadratic system (2.5) represents a conic-conic intersection problem between the two conics $S_1(X,Y) = 0$ and $S_2(X,Y) = 0$. One way to solve this intersection problem is to identify an $S_3 = aS_1 + bS_2$ that eliminates X^2 (or Y^2) term. Then S_3 is linear in X, and can be solved for X in terms of Y. This equation for X is substituted into either S_1 or S_2, yielding a quartic equation in Y. Another way is to consider the conic pencil $S_3 = S_1 + \mu S_2$ and identify the parameter μ that makes S_3 a degenerate conic. This involves solving a cubic instead of a quartic equation in μ, of which we need only one real root. S_3 is then decomposed into one or two lines, and the lines are then intersected with either S_1 or S_2. Details can be found, for example, in Art. 187 in [113]. Both methods were used in Fidkowski's implementation [49], and do not differ in speed when implemented in *double* precision. However, in adaptive precision, the second method is preferred as lower-degree polynomials are involved. Note we attempt to intersect S_3 with S_1 or S_2 only if S_3 intersects with the reference triangle; this is easily verified because S_3 represents straight line(s). Further, the choice of S_1 or S_2 to intersect with S_3 depends on the magnitude of μ. For example, when μ is very small, S_3 will be nearly identical to S_1, and intersecting them may require a large amount of precision refinement.

Root-Finding for Cubic Equations

The most expensive step in solving the conic-conic intersection problem lies in solving the cubic equation of μ. While there are analytical formula for roots of cubic equations (see for example [60]), the formula for the case with three distinct roots involves complex numbers or trigonometric functions, which are not available for algebraic numbers. In LEDA, the root for a cubic equation is represented using the *diamond* operator for algebraic numbers [111], which applies Newton's method for root finding whenever precision refinement is needed for sign computation. It is thus critical not to query a duplicate or nearly-duplicate root for performance concerns. We can derive expressions that relate the distance between two roots to the equation coefficients, and thus identify and avoid a duplicate or nearly-duplicate root before solving the equation. Furthermore, these expressions involve only the discriminant and invariants of the equation, which are already computed for classifying the cubic equation. Details are provided in Appendix A.4.

Validity of *oned* Objects

When *oned* objects along background tetrahedron edges are constructed (see *oned*$_3$ and *oned*$_4$ in Figure 2-1(b)), we need to determine their validity, where being valid means inside the computational domain or on its boundary. This information can be achieved by evaluating the inward patch normal at the intersecting *zerod* object (*zerod*$_4$ in Figure 2-1(b)). This evaluation is a very expensive step because the coordinates of the *zerod* objects are the roots of (2.2) and hence have a complex construction DAG. Instead of performing this evaluation for every *oned* object, we evaluate the validity of only one using this method, and the validity of very other background-edge *oned* object can be deduced based on topology by traversing through each background edge. Whenever an intersection point with an odd multiplicity is encountered when traversing, the validity of the next *oned* object is switched from that of the current one. For instance, traversing from *oned*$_3$ to *oned*$_4$ in Figure 2-1(b) encounters an intersection point *zerod*$_4$, and so these two *oned* objects must be on different sides of the quadratic-patch geometry.

We also determine the validity of *oned* objects on patch edges and faces based on the same principle when using the adaptive precision arithmetic. We rely on topology information whenever possible instead of computing based on the coordinates of intersection points, which can involve a large construction DAG. Construction of each type of *oned* object is described in detail in Appendix B.2.

2.5 Results for Robustness and Efficiency

With all the techniques implemented for efficiency improvement, the testing results for the 15 similar background meshes intersecting with the ONERA M6 geometry is summarized in Table 2.1. The timing results represent the average time of intersecting the 15 meshes (or the meshes that return a valid cut mesh in the case of using *double* precision). The solution time is for an inviscid transonic flow on the same mesh using the ProjectX DG solver [13, 43, 51]. The DG discretization has a polynomial degree of $p = 1$, and the nonlinear solver starts from a converged flow solution on a similar mesh. The flow has a freestream Mach number of 0.8395, an angle of attack of 3.06°, and a sideslip angle of 0°. With the efficiency improvement, the intersection time represents only a small fraction of the solution time on these coarse meshes. Furthermore, the intersection code is parallelized by partitioning the background mesh, and almost a linear speedup is observed. Note that the EGC guarantees the consistency between cut topologies across partitions, and hence ensures a correct parallelized intersection algorithm. Details of the parallelization implementation are provided in Appendix B.4.

Table 2.1: Correctness and performance for cut-cell intersection

Type	Correctness	Approximate fraction of solution time
double precision	47%	0.1%
Adaptive precision	100%	400%
Adaptive precision with efficiency improvement	100%	5%

Chapter 3

High-Quality Quadrature Rule for Cut Cells

3.1 Introduction

As a cut-cell mesh can have arbitrarily shaped elements, a quadrature rule for each of these elements and their faces is required in a finite element discretization. One possible approach is to subdivide each cut element into possibly-curved simplices on which standard quadrature rules can be applied. However, this approach cycles back to the original problem of meshing an arbitrary (curved) domain. A more general quadrature rule is thus needed: find a quadrature rule, $\{\mathbf{x}_q, w_q\}$, such that for an integrand $f(\mathbf{x})$, we have

$$\int_\Omega f(\mathbf{x})d\mathbf{x} \approx \sum_{q=1}^{n_q} w_q f(\mathbf{x}_q), \tag{3.1}$$

where Ω is an arbitrary closed domain in two or three dimensions, and the choice of $\{\mathbf{x}_q, w_q\}$ is independent of the function $f(\mathbf{x})$. Because the whole solution strategy is promised to be fully-automated, the generation of the quadrature rules needs to be achieved in an automated manner. In addition, a high quadrature quality is also required, since lack of integration quality has an adverse impact on the quality of residual (and Jacobian matrix) evaluation for a finite element discretization. Such impact can result in poor nonlinear solver convergence, especially when higher-order polynomial approximation is employed. This will be demonstrated in Section 5.1 for a discontinuous Galerkin discretization applied to aerodynamics problems.

Quadrature rules are typically deigned such that (3.1) is exact for every function in the polynomial space $\mathbb{P}_p^d$, which spans all polynomials of a (total) degree p in d variables. The degree p is often referred to as the algebraic degree of the quadrature rule:

Definition 3.1. *A quadrature rule, $\{\mathbf{x}_q, w_q\}$, has an algebraic degree p if it is exact for all polynomials of degree at most p but not exact for at least one polynomial of degree $p+1$.*

Let $\{\psi_i\}_{i=1}^{n_b}$ denote the basis functions of the space $\mathbb{P}_p^d$, where $n_b \equiv \dim \mathbb{P}_p^d$. Then a necessary and sufficient condition for a degree-p quadrature rule is (3.1) being exact for each ψ_i:

$$\int_\Omega \psi_i(\mathbf{x}) d\mathbf{x} = \sum_{q=1}^{n_q} w_q \psi_i(\mathbf{x}_q), \quad i = 1, ..., n_b. \tag{3.2}$$

Assuming we have means to compute $\int_\Omega \psi_i(\mathbf{x}) d\mathbf{x}$, then (3.2) represents a polynomial system of n_b equations for $(d+1)n_q$ variables: $\{\mathbf{x}_q\}_{q=1}^{n_q}$ and $\{w_q\}_{q=1}^{n_q}$. Note when $n_q \geq n_b$, this system is guaranteed to have solutions with each $\mathbf{x}_q$ inside Ω and each w_q non-negative [37]. Cools [37, 38] provides excellent reviews on the construction of quadrature rules based on (3.2), and classifies the existing methods into two main approaches:

1. solve the system (3.2) directly, using for example Newton's method;

2. search for quadrature points at which a set of orthogonal polynomials vanish.

The first approach involves root-finding for polynomial systems, and often lacks robustness in converging an iterative solver. This approach is thus usually applied when certain symmetry structure exists for the integration domain and can reduce the size of the system (3.2); see for example [132] where quadrature rules are derived using this approach on triangles and tetrahedra. Mousavi *et al.* [89] pursued this route for arbitrary polygons without symmetry structures, but the method requires an initial set of degree-p quadrature points, and the convergence of the Newton's method is very sensitive on the choice of the initial points. For the second approach, it is well known in one dimension that the n roots of a degree-n polynomial that is orthogonal to all lower-degree polynomials lead to a quadrature rule of degree $2n-1$. However, extension of this idea to higher dimensions encounters significant difficulty, even for standard regions such as triangles. One main challenge is to construct a proper set of (multivariate) orthogonal polynomials that has a sufficient number of common roots. A review on the current state of the art and challenges for this approach can be found in [36]. Therefore, neither approach is robust and/or computationally affordable for cut-cell applications, where a quadrature rule needs to be derived for each cut element in a fully-automated manner.

Another route to high-quality quadrature is through approximating the integration region by "canonical" shapes, for which high-quality quadrature rule is available. For example, Modisette recognized certain cut elements in two dimensions through parametric polynomial mapping over triangles and quadrilaterals [87]; Sommariva and Vianello approximated an arbitrarily shaped two-dimensional domain using polynomial splines [114]. However, in addition to the fact that these methods modify the definition of the embedded geometry, their extension to three dimensions is non-trivial. More specifically, when two faces that share a common edge are approximated (by polynomial mappings), it is not easy to ensure the two approximated faces still define a common edge. This can undermine the premise of quadratic-patch representation being watertight. The focus of this chapter is thus on deriving quadrature rules for arbitrarily shaped elements without appealing to the "canonical" shapes. Another method of approximating an integration domain is proposed by Natarajan *et al.*, where an arbitrary polygon is mapped into a unit circle through conformal mapping [91];

however, this method does not have an obvious extension to three dimensions either.

The objective of this chapter is to develop an algorithm that generates a high-quality quadrature rule in an automated manner for each arbitrarily shaped element in two and three dimensions. We first explore proper metrics for evaluating quadrature quality, and propose two criteria in Section 3.2 for the cut-cell quadrature rule to satisfy: algebraic degree and a defined quadrature quality measure. The quadrature weights are computed to fulfill the criterion on algebraic degree, and the quadrature points are selected to improve the quadrature quality as presented in Sections 3.3. Section 3.4 describes how the integration of basis polynomials is evaluated, and numerical examples are presented in Section 3.5. In Section 3.6, we summarize the entire algorithm for generating the cut-cell quadrature rule. The proposed algorithm does not rely upon any symmetry information or geometry approximation, and does not involve high-order polynomial root-finding. Note, without loss of generality, we always assume the integration domain has a unit volume in this chapter for presentation brevity.

3.2 Criteria for Cut-Cell Quadrature Rule

Before attempting to construct a quadrature rule for cut cells, we need to first define metrics for assessing the quality of quadrature rules. In this section, we define two such metrics, and propose two corresponding criteria for the cut-cell quadrature rule to satisfy.

3.2.1 Algebraic Degree

A standard measure for quadrature quality is the algebraic degree p defined in Definition 3.1. The first criterion is thus:

Criterion 3.2. *The cut-cell quadrature rule has a user-specified algebraic degree p, and so satisfies Eq.* (3.2), *or written in matrix form:*

$$\boldsymbol{V}\mathbf{w} = \mathbf{b}, \tag{3.3}$$

where $\mathbf{w} \in \mathbb{R}^{n_q}$ *is the vector of quadrature weights,* $\boldsymbol{V} \in \mathbb{R}^{n_b \times n_q}$ *is the Vandermonde matrix on quadrature points with* $\boldsymbol{V}_{iq} = \psi_i(\mathbf{x}_q)$*, and* $\mathbf{b} \in \mathbb{R}^{n_b}$ *has* $\mathbf{b}_i = \int_\Omega \psi_i(\mathbf{x})d\mathbf{x}$*.*

In this work, we achieve this criterion through manipulating only $\{w_q\}$, while $\{\mathbf{x}_q\}$ will be chosen based on the second criterion we propose. More specifically, given a set of n_q points, where $n_q \geq n_b$, we define $\{w_q\}$ through a projection of the integrand $f(\mathbf{x})$ onto the polynomial space $\mathbb{P}_p^d$: find $\mathbf{F} \in \mathbb{R}^{n_b}$ such that

$$\mathbf{F} = \arg\min_{\mathbf{F}} \sum_{q=1}^{n_q} c_q \left(\sum_{i=1}^{n_b} \mathbf{F}_i \psi_i(\mathbf{x}_q) - f(\mathbf{x}_q) \right)^2, \tag{3.4}$$

where c_q is the (approximate) volume of the Voronoi cell around $\mathbf{x}_q$. This weighted least-squares problem has a solution of

$$\mathbf{F} = \left(\boldsymbol{V}\mathbf{C}\boldsymbol{V}^T\right)^{-1} \boldsymbol{V}\mathbf{C}\mathbf{f}, \tag{3.5}$$

where $\mathbf{C} \in \mathbb{R}^{n_q \times n_q}$ is a diagonal matrix with entries being c_q, and $\mathbf{f} \in \mathbb{R}^{n_q}$ has $\mathbf{f}_q = f(\mathbf{x}_q)$. The integral of the function $f(\mathbf{x})$ is then approximated by the integral of the projected polynomial:

$$\int_\Omega f(\mathbf{x})d\mathbf{x} \approx \int_\Omega \sum_{i=1}^{n_b} \mathbf{F}_i \psi_i(\mathbf{x})d\mathbf{x} = \mathbf{F}^T\mathbf{b} = \mathbf{f}^T\mathbf{C}\boldsymbol{V}^T \left(\boldsymbol{V}\mathbf{C}\boldsymbol{V}^T\right)^{-1} \mathbf{b},$$

and we can define the quadrature weights as

$$\mathbf{w} = \mathbf{C}\boldsymbol{V}^T \left(\boldsymbol{V}\mathbf{C}\boldsymbol{V}^T\right)^{-1} \mathbf{b}, \tag{3.6}$$

which is independent of the integrand $f(\mathbf{x})$. If the integrand $f(\mathbf{x})$ belongs to the polynomial space $\mathbb{P}_p^d$, the projected polynomial will be equal to the integrand due to (3.4), i.e. $f(\mathbf{x}) = \sum_{i=1}^{n_b} \mathbf{F}_i \phi_i(\mathbf{x})$, making (3.3) satisfied. Note this approach is generalized from the derivation of quadrature weights in Fidkowski's work [49], where c_q was unity for every quadrature point.

One of the reasons that the algebraic degree has been a standard measure for quadrature quality is that a smooth function $f(\mathbf{x})$ has exponentially decaying spectral expansion coefficients. By capturing the integral of its first p spectral terms, the quadrature rule is expected to capture the integral of the function itself. Hence increasing the quadrature degree p should lead to an exponential convergence of quadrature error. However, one important aspect is overlooked in this reasoning. Although the quadrature rule integrates exactly the first p terms, the rule may magnify the higher-degree terms not captured, and may produce large errors that pollute the result. A similar argument is also demonstrated in the work of Trefethen [119], which shows that Clenshaw-Curtis quadrature is as competitive as Gauss quadrature even though it has a lower degree for the same number of points. Thus, merely increasing quadrature degree is not sufficient for a high-quality integration.

3.2.2 Quality Measure

For a general quadrature rule $\{\mathbf{x}_q, w_q\}_{q=1}^{n_q}$, we define the quantity:

$$Q \equiv \mathbf{w}^T \mathbf{C}^{-1} \mathbf{w}, \tag{3.7}$$

where $\mathbf{w}$ is the vector of quadrature weights, and $\mathbf{C}$ is defined as in (3.5). In this section, we first characterize the quadrature rule defined by (3.6) and discuss the properties of Q, and then demonstrate Q can be a measure of quadrature quality for this rule.

Let X_{n_q} represent a set of quadrature points $\{\mathbf{x}_q\}_{q=1}^{n_q}$, and let $\{X_{n_q}\}_{n_q=1}^{\infty}$ represent a sequence of such sets. For a sequence that satisfies Assumption 3.3 given below, Theorem 3.4 characterizes the asymptotic behavior of the quadrature rule (3.6). The proof is provided in Appendix C.1.

Assumption 3.3. *Let $f(\mathbf{x})$ be a Riemman integrable function. For the sequence*

$\{X_{n_q}\}_{n_q=1}^{\infty}$ *of quadrature point sets* $X_{n_q} \equiv \{\mathbf{x}_q\}_{q=1}^{n_q}$, *we assume*

$$\lim_{n_q \to \infty} \sum_{q=1}^{n_q} c_q f(\mathbf{x}_q) = \int_{\Omega} f(\mathbf{x}) d\mathbf{x},$$

where c_q *is the volume of the Voronoi cell around* $\mathbf{x}_q$. *We also assume a convergence rate* $r > 0$*:*

$$\left| \sum_{q=1}^{n_q} c_q f(\mathbf{x}_q) - \int_{\Omega} f(\mathbf{x}) d\mathbf{x} \right| \sim \mathcal{O}\left(n_q^{-r}\right). \tag{3.8}$$

Theorem 3.4. *Let a set of quadrature points* $\{\mathbf{x}_q\}_{q=1}^{n_q}$ *belong to a sequence* $\{X_{n_q}\}$ *satisfying Assumption 3.3, and let the quadrature weights* $\{w_q\}_{q=1}^{n_q}$ *be defined by* (3.6). *Then we have the following:*

1. *the quadrature error converges to zero:*

$$\lim_{n_q \to \infty} \sum_{q=1}^{n_q} w_q f(\mathbf{x}_q) = \int_{\Omega} f(\mathbf{x}) d\mathbf{x}; \tag{3.9}$$

2. *Q defined by* (3.7) *converges to one from above:*

$$Q > 1, \quad \text{and} \lim_{n_q \to \infty} Q = 1; \tag{3.10}$$

3. *the quadrature weight for each point converges to the volume of its Voronoi cell:*

$$\lim_{n_q \to \infty} w_q = c_q, \quad \forall q. \tag{3.11}$$

Wilson [123] and Huybrechs [63] also derived the quadrature weights $\{w_q\}$ in (3.6) but from a different perspective, which will be presented later in Eq. (3.12). Both authors discussed Item 3 of Theorem 3.4 in more detail. In particular, if $\{c_q\}$ is chosen such that the convergence rate r in (3.8) is higher, the quadrature rule defined by $\{w_q\}$ will also have the same higher rate for the convergence in (3.9). For uniformly spaced points, with c_q being the volume of Voronoi cell, we have $r = \frac{2}{d}$ (for a sufficiently smooth integrand); and the convergence rate for Q in (3.10) is observed to be $2r$. A numerical example defining $\{c_q\}$ by Simpson's rule in one dimension is demonstrated in [63]. Although the quadrature rules defined by $\{w_q\}$ and $\{c_q\}$ have the same convergence rate with respect to n_q, the weight $\{w_q\}$ gives an algebraic degree p regardless the choice of $\{c_q\}$, and is interpreted as a higher-degree correction to $\{c_q\}$ in [63, 123].

For any degree-p quadrature rule, we can bound the quadrature error using Q as stated in the following theorem. The proof follows the proof for Theorem 4.1 in [119], and is provided in Appendix C.1.

Theorem 3.5. *For any degree-p quadrature rule, $\{\mathbf{x}_q, w_q\}_{q=1}^{n_q}$, where $p \geq 0$, we have*

$$\left| \sum_{q=1}^{n_q} w_q f(\mathbf{x}_q) - \int_\Omega f(\mathbf{x}) d\mathbf{x} \right| \leq C(1 + \sqrt{Q}),$$

where Q is defined in (3.7), and C is a constant based on the integrand $f(\mathbf{x})$.

For the quadrature rule defined by (3.6), it is conjectured from numerical evidence that a tighter bound in terms of $\mathcal{Q} \equiv \sqrt{Q-1}$ exists for the quadrature error. Figure 3-1 shows one example, where we generate 1000 fifth-degree quadrature rules on the domain $[0, 1]$. Each rule has $n_q = 15n_b$ quadrature points randomly sampled from a uniform distribution, and quadrature weights computed from (3.6). We then apply each rule to integrate a smooth function $\sin(\pi x)$, for which the spectral expansion terms decay exponentially, and the quadrature error can reflect whether the rule magnifies the higher-degree spectral terms. Figure 3-1 plots the quadrature error versus $\mathcal{Q}$, and a strong correlation between the error (or its upper bound) and $\mathcal{Q}$ is observed. As we also observe the quadrature error and $\mathcal{Q}$ converge to zero at the same rate as n_q increases, we expect a similar correlation for a higher n_q as well. Further, even though each rule in Figure 3-1 has the same algebraic degree, the error upper bound varies almost three orders of magnitude. Therefore, in addition to achieving a specified algebraic degree, we propose the second criterion for the cut-cell quadrature rule:

Criterion 3.6. *The cut-cell quadrature rule needs to have a value of Q less than a user-specified $Q^{threshold}$.*

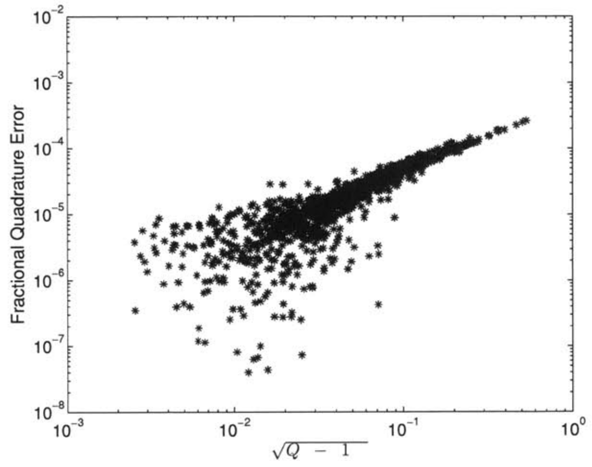


Figure 3-1: Quadrature error vs. $\sqrt{Q-1}$

As Q is a reasonable measure of quadrature quality, designing a rule with minimum Q may be desirable. Such a rule corresponds to the solution of an optimization formulation: given $\{\mathbf{x}_q\}_{q=1}^{n_q}$, where $n_q \geq n_b$, find

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathbf{w}^T \mathbf{C}^{-1} \mathbf{w}, \text{ subject to } \boldsymbol{V}\mathbf{w} = \mathbf{b}, \tag{3.12}$$

where $\mathbf{C}$ is defined in (3.5), and $\boldsymbol{V}$ and $\mathbf{b}$ are defined in (3.3). This optimization formulation is also proposed in [63, 123], and has a unique closed-form solution. However, the optimal solution $\mathbf{w}^*$ is in fact identical to the quadrature weights defined by (3.6). This suggests that a decrease in Q has to be achieved in the choice of $\{\mathbf{x}_q\}$, which will be discussed in Section 3.3.

Another common measure for quadrature quality is the positivity of quadrature weights. A positive quadrature rule is one with all positive weights, and its existence for an arbitrary domain was first proven by Tchakloff [118] when $n_q = n_b$. Davis later provided another proof by proposing a construction method of such a rule [39], but the method serves more as a mathematical proof than a practical construction algorithm. Huybrechs [63] proposed an approach using non-negative least squares on a large number of sample points. However, when the number of sample points is not sufficient, the approach can lead to a (positive) rule that violates (3.2) and does not even integrate a constant function exactly. Note that the quadrature weights defined by (3.6) are in fact also positive for a sufficiently large n_q due to Item 3 in Theorem 3.4.

3.3 Quadrature Point Selection

Given a set of quadrature points $\{\mathbf{x}_q\}$, let the quadrature weights be computed from (3.6), then Q defined in (3.7) is equal to

$$Q(\{\mathbf{x}_q\}) = \mathbf{b}^T(\boldsymbol{V}\mathbf{C}\boldsymbol{V}^T)^{-1}\mathbf{b}, \tag{3.13}$$

which is independent of the chosen polynomial basis. For a given basis, $\mathbf{b}$ is independent of $\{\mathbf{x}_q\}$, and so Q in (3.13) decreases in general if $\{\mathbf{x}_q\}$ is selected such that the matrix $\boldsymbol{V}\mathbf{C}\boldsymbol{V}^T$ is better conditioned. While each column in $\boldsymbol{V}$ is for one quadrature point and independent of other points, each entry c_q in $\mathbf{C}$ corresponds to the volume of the Voronoi cell around $\mathbf{x}_q$ and relies on the choice of the entire set $\{\mathbf{x}_q\}$. We thus consider a simpler problem of choosing $\{\mathbf{x}_q\}$ to improve the conditioning of the matrix $\boldsymbol{V}\boldsymbol{V}^T$ instead. A natural solution to this problem is the Fekete points, which consist of a set of n_b points that maximizes the absolute value of the determinant of $\boldsymbol{V}$. However, they are known analytically in only very few instances, for example in a circle or a cube. Taylor *et al.* proposed a steepest ascent algorithm to find the Fekete points in a triangle [117], but the method is not affordable if we need to find a different quadrature rule for every cut element. In this work, we use the empirical interpolation points developed by Barrault *et al.* [11], which are applied to polynomial interpolation by Maday *et al.* [83] and named the magic points. In the rest of this section, we first briefly review the construction of the magic points, and then prove they are asymptotically equivalent to the Fekete points under certain conditions.

Let $\mathcal{U}$ denote a set of functions in $L^\infty(\Omega)$, where the cardinality of $\mathcal{U}$ can be infinite.

The objective of the magic point construction is to find a space $X_M \subset L^\infty(\Omega)$ and a set of interpolation points $\{\mathbf{x}_i\}_{i=1}^M$, such that the interpolation error $\|u - \mathcal{I}_M[u]\|_{L^\infty(\Omega)}$ is small for any $u \in \mathcal{U}$, where $\dim(X_M) = M$, and the interpolant $\mathcal{I}_M : \mathcal{U} \to X_M$ is defined by

$$\mathcal{I}_M[u](\mathbf{x}_i) = u(\mathbf{x}_i), \quad i = 1, ..., M.$$

This is achieved through a greedy procedure. We first define u_1 as

$$u_1 = \arg\max_{u \in \mathcal{U}} \|u\|_{L^\infty(\Omega)},$$

and define the first interpolation point by

$$\mathbf{x}_1 = \arg\max_{\mathbf{x} \in \overline{\Omega}} |u_1(\mathbf{x})|.$$

Then suppose that $\{\mathbf{x}_i\}_{i=1}^M$ has been chosen, the next interpolation point is determined according to:

$$u_{M+1} = \arg\max_{u \in \mathcal{U}} \|u - \mathcal{I}_M[u]\|_{L^\infty(\Omega)}, \tag{3.14}$$

$$\mathbf{x}_{M+1} = \arg\max_{\mathbf{x} \in \overline{\Omega}} |u_{M+1}(\mathbf{x}) - \mathcal{I}_M[u_{M+1}](\mathbf{x})|. \tag{3.15}$$

In implementation, the set $\mathcal{U}$ is replaced by a set of monomials with a total degree up to a specified degree n, denoted by W_n; and the domain $\overline{\Omega}$ is discretized using a large number of (uniformly spaced) sample points, denoted by S. The optimization problems (3.14) and (3.15) are then approximated by

$$u_{M+1} = \arg\max_{u \in W_n} \|u - \mathcal{I}_M[u]\|_{L^\infty(\Omega)} \tag{3.16}$$

$$\mathbf{x}_{M+1} = \arg\max_{\mathbf{x} \in S} |u_{M+1}(\mathbf{x}) - \mathcal{I}_M[u_{M+1}](\mathbf{x})|, \tag{3.17}$$

which are solved using an exhaustive search. The interpolant $\mathcal{I}_M$ is constructed by solving for the coefficients $\{\beta_j\}_{j=1}^M$ from

$$\mathcal{I}_M[u] \equiv \sum_{j=1}^{M} \beta_j q_j(\mathbf{x}_i) = u(\mathbf{x}_i), \quad i = 1, ..., M, \tag{3.18}$$

where the interpolation basis q_j is defined during the greedy process of choosing u_i and $\mathbf{x}_i$:

$$\begin{aligned} q_1 &\equiv \frac{u_1}{u_1(\mathbf{x}_1)}, \\ q_{M+1} &\equiv \frac{u_{M+1} - \mathcal{I}_M[u_{M+1}]}{u_{M+1}(\mathbf{x}_{M+1}) - \mathcal{I}_M[u_{M+1}](\mathbf{x}_{M+1})}, \quad M \geq 1. \end{aligned}$$

Note the linear system (3.18) is lower triangular with unity diagonal (and hence invertible) as proven in [83].

The point selection based on (3.15) is in fact equivalent to maximizing the absolute value of the Vandermonde determinant, as stated in Lemma 3.7. We then present the asymptotic behavior of the magic points in Theorem 3.9, assuming the set of sample points S belongs to a weakly admissible mesh (WAM) defined in Definition 3.8; see [22] for more properties of a WAM.

Lemma 3.7. *The optimization problem* (3.15) *is equivalent to*

$$\mathbf{x}_{M+1} = \arg\max_{\mathbf{x}\in\overline{\Omega}} \left|\det(\boldsymbol{V}_{\{u_i\}_{i=1}^{M+1}}(\mathbf{x}_1, ..., \mathbf{x}_M, \mathbf{x}))\right| \tag{3.19}$$

where $\boldsymbol{V}_{\{u_i\}_{i=1}^{M+1}}(\mathbf{x}_1, ...\mathbf{x}_M, \mathbf{x}) \in \mathbb{R}^{(M+1)\times(M+1)}$ *is the Vandermonde matrix of the functions* $\{u_i\}_{i=1}^{M+1}$ *evaluated at the points* $\{\mathbf{x}_i\}_{i=1}^M \cup \mathbf{x}$.

Proof. From (3.18), there exists $\{\alpha_j\}_{j=1}^M$ such that

$$\mathcal{I}_M[u_{M+1}](\mathbf{x}_i) = \sum_{j=1}^{M} \alpha_j u_j(\mathbf{x}_i) = u_{M+1}(\mathbf{x}_i), \quad i = 1, ..., M. \tag{3.20}$$

This is because span$\{q_1, ..., q_M\}$ coincide with span$\{u_1, ..., u_M\}$ as proven in [83]. We then have for any $\mathbf{x} \in \overline{\Omega}$:

$$
\begin{aligned}
&\left|\det(\boldsymbol{V}_{\{u_i\}_{i=1}^{M+1}}(\mathbf{x}_1, ..., \mathbf{x}_M, \mathbf{x}))\right| \\
&= \left|\det \begin{pmatrix} u_1(\mathbf{x}_1) & \cdots & u_M(\mathbf{x}_1) & u_{M+1}(\mathbf{x}_1) \\ \vdots & & \vdots & \vdots \\ u_1(\mathbf{x}_M) & \cdots & u_M(\mathbf{x}_M) & u_{M+1}(\mathbf{x}_M) \\ u_1(\mathbf{x}) & \cdots & u_M(\mathbf{x}) & u_{M+1}(\mathbf{x}) \end{pmatrix}\right| \\
&= \left|\det \begin{pmatrix} u_1(\mathbf{x}_1) & \cdots & u_M(\mathbf{x}_1) & \cancelto{0}{u_{M+1}(\mathbf{x}_1) - \sum_{j=1}^M \alpha_j u_j(\mathbf{x}_1)} \\ \vdots & & \vdots & \vdots \\ u_1(\mathbf{x}_M) & \cdots & u_M(\mathbf{x}_M) & \cancelto{0}{u_{M+1}(\mathbf{x}_M) - \sum_{j=1}^M \alpha_j u_j(\mathbf{x}_M)} \\ u_1(\mathbf{x}) & \cdots & u_M(\mathbf{x}) & u_{M+1}(\mathbf{x}) - \sum_{j=1}^M \alpha_j u_j(\mathbf{x}) \end{pmatrix}\right| \\
&= \left|\det \begin{pmatrix} u_1(\mathbf{x}_1) & \cdots & u_M(\mathbf{x}_1) \\ \vdots & & \vdots \\ u_1(\mathbf{x}_M) & \cdots & u_M(\mathbf{x}_M) \end{pmatrix}\right| \cdot \left|u_{M+1}(\mathbf{x}) - \mathcal{I}_M[u_{M+1}](\mathbf{x})\right|,
\end{aligned}
$$

where the first factor is equal to $\left|\det(\boldsymbol{V}_{\{u_i\}_{i=1}^{M}}(\mathbf{x}_1, ..., \mathbf{x}_M))\right|$, which is independent of $\mathbf{x}$. Therefore, the objective functions in (3.15) and (3.19) are the same up to a constant factor, and so lead to the same point $\mathbf{x}_{M+1}$. □

Definition 3.8. *Let S_n denote a set of points in $\overline{\Omega}$, then a weakly admissible mesh (WAM) is a sequence of such sets, $\{S_n\}_{n=1}^{\infty}$, such that*

$$||p||_{L^\infty(\Omega)} \leq C(S_n)||p||_{L^\infty(S_n)}, \quad \forall p \in \mathbb{P}_n^d,$$

where both $C(S_n)$ and $|S_n|$ grow at most polynomially with n.

Theorem 3.9. *Let $W_n \equiv \{w_1, ..., w_N\}$ represent a set of monomials in the order of increasing total degree up to n, i.e. deg(w_i) $\leq$ deg(w_j) $\leq n$ for $i \leq j$. Let S_n denote a set of points that belongs to a WAM defined in Definition 3.8. If we let the function choice in* (3.16) *be simply $u_{M+1} = w_{M+1}$, then the points selected from S_n using* (3.17) *will asymptotically lead to the Fekete points as $n \to \infty$.*

Proof. Based on Lemma 3.7, the point selection from (3.17) is identical to that from (3.19) with $\overline{\Omega}$ being discretized by the same set of sample points S_n. Further, the optimization problem (3.19) is in fact the same as the definition for the discrete Leja points presented in [23], which are proven to have the same asymptotic distribution as the Fekete points when W_n and S_n satisfy the conditions stated in the theorem. $\square$

3.4 Integration of Basis Polynomials

When we compute the quadrature weights from (3.6), we need to evaluate the vector $\mathbf{b}$, which is the integration of the basis polynomials ψ_i's in Ω. We follow the idea in [49], that is, for each ψ_i, we define a vector function G_{ik}, $k = 1, ..., d$, such that $\sum_k \partial_k G_{ik} = \psi_i$. Then we apply the divergence theorem:

$$\int_{\Omega} \psi_i d\mathbf{x} = \oint_{\partial\Omega} \sum_{k=1}^{d} G_{ik} \hat{\mathbf{n}}_k dS, \tag{3.21}$$

where the surface integral is evaluated using the quadrature rule in the dimension $d-1$. In this work, G_{ik} is defined on the oriented bounding box of Ω. For presentation brevity, assume the bounding box is axis aligned: $x_k \in [x_k^{\min}, x_k^{\max}]$, then G_{ik} is defined as

$$G_{ik} = (x_k - x_k^{\min}) \prod_{j=1}^{d} \phi_{i_j} \left(\frac{x_k - x_k^{\min}}{x_k^{\max} - x_k^{\min}} \right), \quad k = 1, ..., d,$$

where each ϕ_{i_j} is the Legendre polynomials defined on $[0, 1]$, and its total degree is smaller than (or equal to) the degree of the quadrature rule, i.e. $\sum_j \deg(\phi_{i_j}) \leq p$.

When we evaluate (3.21) on the surface of quadratic patches, the term $\oint_{\partial\kappa}(\cdot)dS$ involves integration (of polynomials) along conics, which requires a parameterization of conics. An overview of different conic parameterizations can be found in Chapter 7 of [101]. In this work, we choose the one with the property that a point sequence from evenly spaced parameters forms a polygon that encloses the maximum inscribed area [101]. However, this parameterization can be very sensitive to precision when the conic shape is close to being a straight line. For such a conic, a polar representation [88] with respect to a carefully chosen origin is used. More details about these two parameterizations are provided in Appendix C.2.

3.5 Numerical Examples

In this section, we demonstrate two examples where the quadrature points are selected based on (3.16) and (3.17), and the quadrature weights are computed from (3.6) for a specified degree p. The set W_n in (3.16) initially consists of monomials with a degree up to $n = p$. Let $\{u_i\}_{i=1}^{M}$ denote the M monomials that have been selected based on (3.16). As M increases, the space spanned by $\{u_i\}_{i=1}^{M}$ becomes larger, i.e. $\operatorname{span}\left(\{u_i\}_{i=1}^{M}\right) \supset \operatorname{span}\left(\{u_i\}_{i=1}^{M-1}\right)$, and the objective function in (3.16), $\|u - \mathcal{I}_M[u]\|_{L^\infty(\Omega)}$, in general decreases for any $u \in W_n$. When $M \approx |W_n|$, this objective function can be (almost) machine precision for any $u \in W_n$, making round-off error affect the choice of u_{M+1} from (3.16). Therefore, we should always keep $|W_n|$ sufficiently large compared to M. In this work, when $M \geq 0.75|W_n|$, we enlarge the set W_n by increasing n by one. The set S in (3.17) consists of a large number of uniformly spaced points inside the integration domain. In this section, we intentionally do not have any sample point on the domain boundary, because for a cut element, the sample points are generated based on its bounding box, and are in general not on the element boundary.

Example 1: "Crown" Shape in Two Dimensions

In this example, we consider a non-convex domain of a "crown" shape as shown in Figure 3-2. In the figure, we also show 150 points selected using (3.17) from a sample of about 4000 uniformly spaced points. After each point is added, we generate a fifth-degree quadrature rule, and integrate a smooth function $f(\mathbf{x}) = \sin(\pi x)\sin(\pi y)$ in the domain. Figure 3-3 shows the quadrature quality measure Q and the quadrature error, both of which decrease with the number of quadrature points n_q. The rate of decrease for $\sqrt{Q-1}$ and (the upper bound of) the error is about the same. Furthermore, for comparison, we also include the results from using uniformly spaced points, which lead to a poorer quadrature quality both in terms of the measure Q and the error.

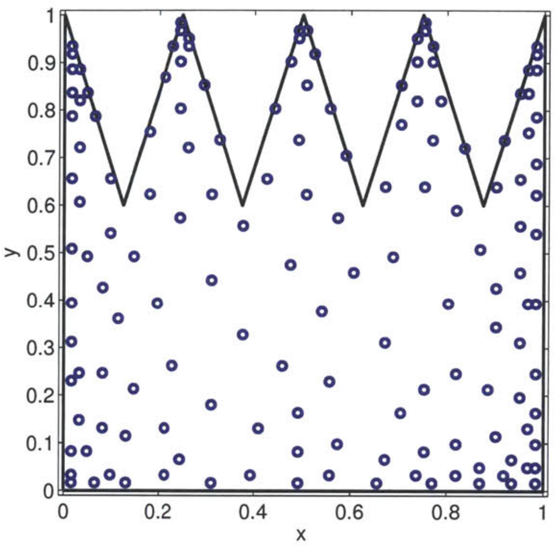
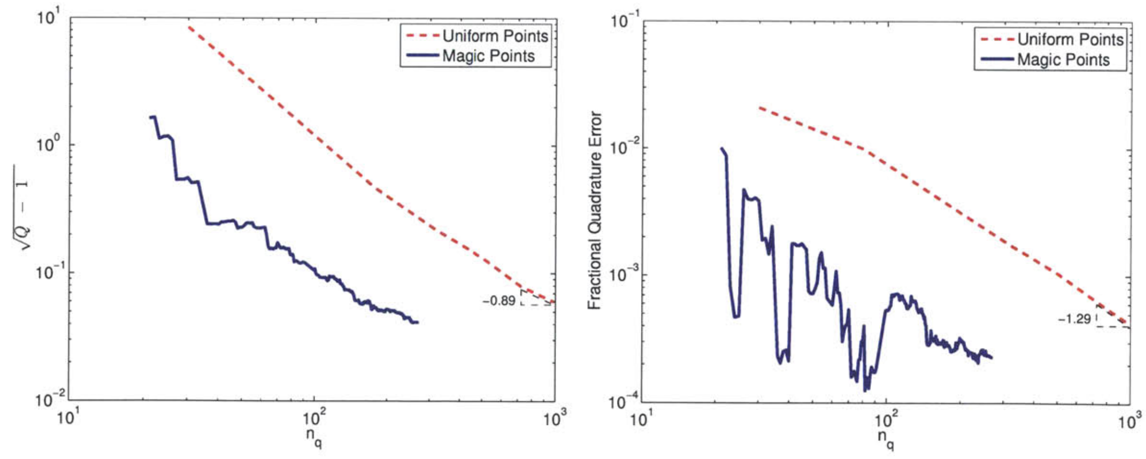


Figure 3-2: "Crown" shape in two dimensions, with 150 points selected from about 4000 points using (3.17)



(a) Quality measure, $\sqrt{Q-1}$ (b) Quadrature error

Figure 3-3: Quadrature quality measure and error for the "crown" shape

Example 2: Curved Element in Three Dimensions

This example has an integration region enclosed by the three coordinate planes and the curved surface $z = -x^2 - y^2 + 1$ as shown in Figure 3-4. The requested quadrature degree is $p = 6$, and the number of basis functions is $n_b = \frac{1}{6}(p+1)(p+2)(p+3)$. The number of sample points for (3.17) is $100n_b$, and the integrand is $\sin(\pi x)\sin(\pi y)\sin(\pi z)$. Figure 3-5 shows the decrease of Q and the error with n_q.

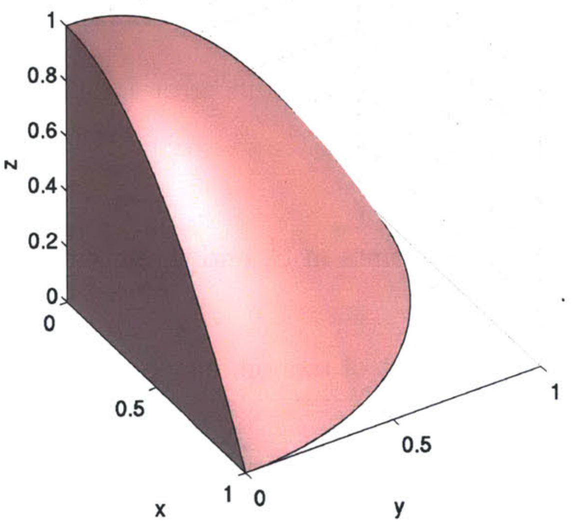
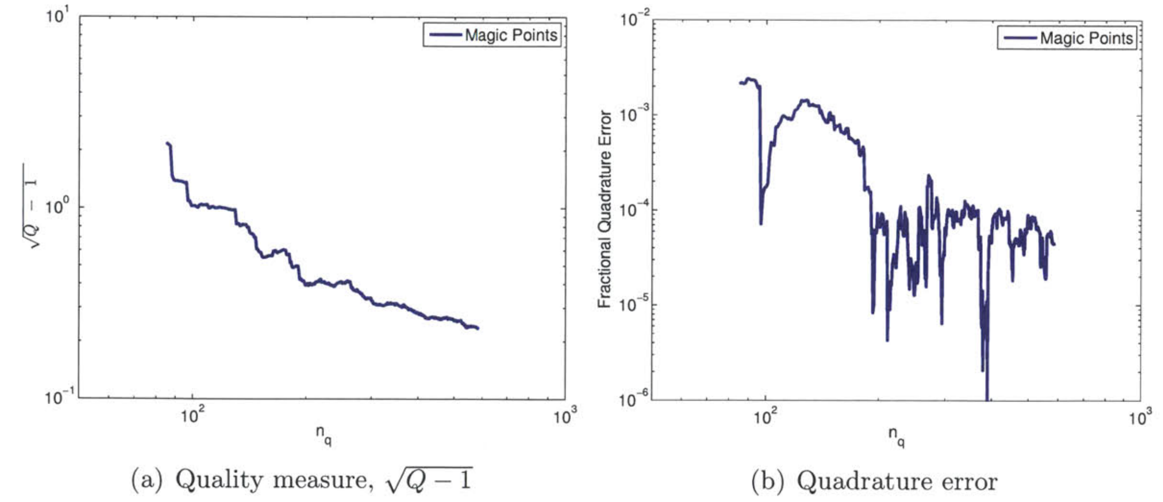


Figure 3-4: Curved element in three dimensions

3.6 Summary: Cut-Cell Quadrature Rule

This section presents the entire algorithm for generating the cut-cell quadrature rule in two and three dimensions. As input, we are given a specified (total) degree p, and a quality threshold, $Q^{\text{threshold}}$, which is set to 5 in this work. Define $n_b \equiv \dim \mathcal{P}_p^d$, and let Ω denote an arbitrary cut element, and $\mathcal{B}$, its oriented bounding box. Denote the set of quadrature points by X, which is initially empty. The algorithm is given as follows:

1. Generate a set S of uniformly spaced sample points in $\overline{\Omega}$, where $|S| \approx 100n_b$. This



(a) Quality measure, $\sqrt{Q-1}$ (b) Quadrature error

Figure 3-5: Quadrature quality measure and error for the curved element in three dimensions

is achieved by populating points in $\overline{\mathcal{B}}$, and those outside $\overline{\Omega}$ are discarded.

2. Let $n = p$, and let W_n consist of monomials up to degree n.

3. Select one point $\mathbf{x}$ from S based on the formulation (3.16) and (3.17), and add $\mathbf{x}$ to X.

4. If $|X| > 0.75|W_n|$, increase n by one, and enlarge W_n to include the new monomials.

5. If $|X| \geq n_b$, evaluate the quadrature weights $\mathbf{w}$ for the specified degree p from (3.6); the choice of basis functions and their integrations are described in Section 3.4.

6. Compute the quality measure Q from (3.7).

7. If $Q \leq Q^{\text{threshold}}$ or $|X| = |S|$, return X and $\mathbf{w}$ as the quadrature rule; otherwise, go to Step 3.

Note in the implementation of computing $\mathbf{w}$ from (3.6), we perform a QR factorization on the matrix $\sqrt{\mathbf{C}}\boldsymbol{V}^T$, so that we have

$$\mathbf{w} = \sqrt{\mathbf{C}}\mathbf{Q}\mathbf{R}^{-T}\mathbf{b},$$

where $\sqrt{\mathbf{C}}\boldsymbol{V}^T = \mathbf{QR}$. Further, we also need to compute c_q, the volume of the Voronoi cell around the point $\mathbf{x}_q \in X$. This is approximated by counting the number of sample points in S that are closer to $\mathbf{x}_q$ than to any other $\mathbf{x} \in X$, that is, we define a set for $\mathbf{x}_q$:

$$S_{\mathbf{x}_q} \equiv \{\mathbf{s} \in S \mid \|\mathbf{x}_q - \mathbf{s}\| \leq \|\mathbf{x} - \mathbf{s}\|, \ \forall \mathbf{x} \in X\},$$

and c_q is approximated by $|S_{\mathbf{x}_q}|/|S|$.

Chapter 4

Discretization, and Output-Based Error Estimation and Adaptation

In this chapter, we first review the discontinuous Galerkin (DG) method for general conservation laws, and describe the choice of solution spaces for cut cells. We then present the dual-weighted residual method proposed by Becker and Rannacher [18, 19] for output error estimation. The adaptation scheme used in this work is the metric optimization framework proposed by Yano and Darmofal [127], and we extend this framework to handle cut cells.

4.1 Discontinuous Galerkin Method for Conservation Laws

This section reviews the DG method for general conservation laws. Let $\Omega \subset \mathbb{R}^d$ be an open, bounded domain in a $d-$dimensional space, and $I \subset \mathcal{R}^+$ be the time interval of interest. A general time-dependent conservation law in the domain Ω expressed in the strong form is given by

$$\frac{\partial u}{\partial t} + \nabla \cdot \mathcal{F}^i(u, \mathbf{x}, t) - \nabla \cdot \mathcal{F}^v(u, \nabla u, \mathbf{x}, t) = \mathcal{S}(u, \nabla u, \mathbf{x}, t), \quad \forall \mathbf{x} \in \Omega,\ t \in I, \tag{4.1}$$

with the boundary conditions

$$\mathcal{B}(u, \mathcal{F}^v(u, \nabla u, \mathbf{x}, t) \cdot \hat{\mathbf{n}}, x, t; \mathrm{BC}) = 0, \quad \forall \mathbf{x} \in \partial\Omega, \ t \in I, \tag{4.2}$$

where $u(\mathbf{x}, t) : \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}^m$ is the $m-$state solution vector. The inviscid flux $\mathcal{F}^i$, the viscous flux $\mathcal{F}^v$, and the source term $\mathcal{S}$ characterize the governing equations to be solved. In this work, the governing equations considered include the advection-diffusion-reaction equation and the Navier-Stokes equations, both of which have the form of the conservation law given in Eq. (4.1). Definitions of these equations are provided in Appendix E.

Let $\mathcal{T}_h$ be a triangulation of the domain Ω with non-overlapping elements, K, such that $\overline{\Omega} = \bigcup_{K \in \mathcal{T}_h} \overline{K}$. The DG discretization seeks a solution in a finite-dimensional approximation space $V_{h,p}$:

$$V_{h,p} \equiv \{v \in \left(L^2(\Omega)\right)^m : v \circ g_K \in (\mathcal{P}^p(K_{\text{ref}}))^m, \ \forall K \in \mathcal{T}_h\}, \tag{4.3}$$

where $\mathcal{P}^p$ denotes the space of p-th degree polynomials, and g_K denotes the mapping from the reference element K_{ref} to the physical element K. Multiplying Eq. (4.1) by a test function, $v \in V_{h,p}$, and integrating by parts over every element leads to the weak formulation of the conservation law, which reads as follows: find $u_{h,p}(\cdot, t) \in V_{h,p}$ such that

$$\sum_{K \in \mathcal{T}_h} \int_K v \frac{\partial u_{h,p}}{\partial t} + \mathcal{R}_{h,p}(u_{h,p}, v) = 0, \quad \forall v \in V_{h,p}. \tag{4.4}$$

The semi-linear weighted residual $\mathcal{R}_{h,p} : V_{h,p} \times V_{h,p} \to \mathbb{R}$ consists of three terms:

$$\mathcal{R}_{h,p}(u_{h,p}, v) = \mathcal{R}^i_{h,p}(u_{h,p}, v) + \mathcal{R}^v_{h,p}(u_{h,p}, v) + \mathcal{R}^s_{h,p}(u_{h,p}, v), \tag{4.5}$$

where $\mathcal{R}^i_{h,p}$, $\mathcal{R}^v_{h,p}$, $\mathcal{R}^s_{h,p}$ denote the discretizations for the inviscid, viscous, and source terms, respectively. In this work, we use Roe's approximate Riemann solver [106] for the inviscid numerical flux, the second form of Bassi and Rebay (BR2) [17] for the viscous discretization, and a mixed form of Bassi *et al.* [14] for the source term with ∇u dependence, which is asymptotically dual-consistent [92]. Boundary conditions (4.2) are enforced weakly by appropriately setting the numerical fluxes on the domain boundaries. For the Navier-Stokes equations, the boundary treatment follows the work of Oliver [92]. Details of the discretization defined by (4.5) and also the discrete solution strategy are provided in Appendix F. Note the particular treatment for interface problems with material discontinuity will be presented in Chapter 6.

4.2 Finite Element Solution Space for Cut Cells

On a cut-cell mesh, let the background mesh be denoted by $\mathcal{T}_{h,b}$, which does not conform to the embedded geometry and consists of simplices from the mesh generation process. The mesh after the cutting process is named the cut mesh, and denoted by $\mathcal{T}_h$, which consists of elements of arbitrary shapes along the geometry. These elements are permitted in the DG scheme defined by (4.5), which requires only integrations on each element K and its faces ∂K; see Appendix F for details of each term in (4.5). In consequence, we can apply the DG method on the cut mesh $\mathcal{T}_h$.

On a typical (high-order) boundary-conforming mesh, each element mapping g_K in (4.3) is often defined by a polynomial of degree q, i.e. $g_K \in (\mathcal{P}^q(K_{\text{ref}}))^d$. However, such a well-defined mapping g_K from a standard reference element often does not exist for cut elements with arbitrary shapes. For these cut elements, we create a linear shadow element in the physical space, and define the mapping g_K from the reference element to the shadow element. Because the shadow element is linear, g_K defines an affine mapping, and the polynomial basis functions in K_{ref} will remain polynomials in physical coordinates. In this work, the choice of the shadow element for each cut cell K is based on the work of Modisette [87]. More specifically, we have three options as listed below in descending order of preference. The hierarchy of these options is set up in an effort to provide the best overlap between the shadow element and the cut element.

1. If a non-singular polynomial mapping $g_K \in (\mathcal{P}^q(K_{\text{ref}}))^d$ is found for K, then K can be treated as a "canonical" element (e.g. a high-order triangle defined by the mapping g_K), and the shadow element will be the linear portion of K. Note this option has only been implemented for K_{ref} being triangle or quadrilateral in two dimensions.

2. If no such mapping is found, the parent background element K_b will be the shadow element when the cut element accounts for more than 50% volume of K_b; an example is the element A in Figure 4-1.

3. When the first two options are unavailable, the shadow element will be the largest simplex in the oriented bounding box of the cut element; an example is the element B in Figure 4-1.

On the cut mesh, an arbitrarily small volume ratio between two neighbor elements can be produced, i.e. an arbitrarily small cut element can be next to a much larger neighbor. The small volume ratio is detrimental to both solution accuracy and linear system conditioning. A detailed analysis was conducted in [87], which also proposed a merging technique to eliminate the issues caused by small volume ratios. More specifically, let two neighbor elements with a small volume ratio be denoted by K_1 and K_2, then we introduce a new merged element $K_{\text{merged}} = K_1 \cup K_2$ into the cut mesh $\mathcal{T}_h$, and remove K_1 and K_2. For the merged element K_{merged}, the shadow element will be the largest simplex in the oriented bounding box of K_{merged}, i.e. the third from the previous list of shadow element options. This merging process is repeated until the volume ratio between any two neighbor elements is higher than some specified threshold value. In our work, the same merging technique is also employed. Note for interface problems, we allow merging of two elements only if they are in the same sub-domain.

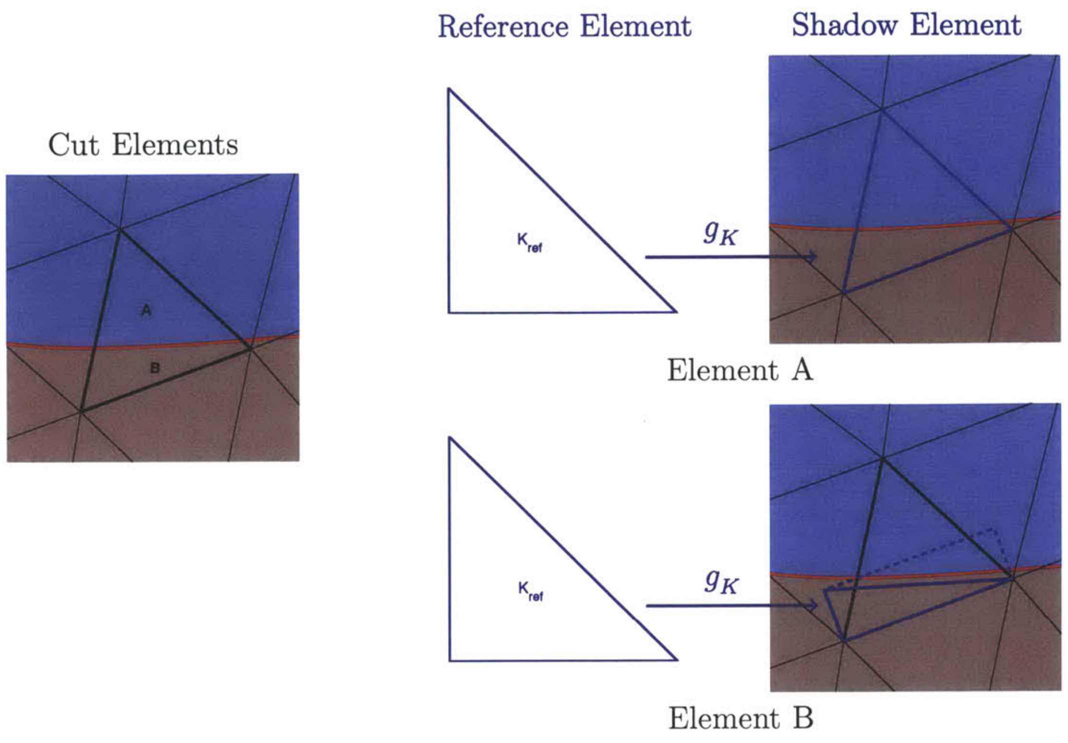


Figure 4-1: Illustration of linear shadow element options assuming a polynomial mapping is not found

4.3 Output Error Estimation

In this work, output error estimation is achieved using the dual-weighted residual (DWR) method proposed by Becker and Rannacher [18, 19]. The method explicitly incorporates the dual problem associated with the output, which links local residuals to the output error. More specifically, let the output of interest be denoted by $J = \mathcal{J}(u)$, where $u \in V$ denotes the exact solution to the governing PDE, and $\mathcal{J}(\cdot) : V \to \mathbb{R}$ is the output functional. Denote the DG solution by $u_{h,p}$ satisfying

$$\mathcal{R}_{h,p}(u_{h,p}, v) = 0, \quad \forall v \in V_{h,p}, \tag{4.6}$$

where $\mathcal{R}_{h,p}$ is the weighted residual from (4.5). The approximate output value computed from $u_{h,p}$ is denoted by $J_{h,p} = \mathcal{J}_{h,p}(u_{h,p})$, where $\mathcal{J}_{h,p}(\cdot) : V_{h,p} \to \mathbb{R}$ is the discrete functional.

In the DWR method, the output error can be expressed as

$$\mathcal{E}_{\text{true}} \equiv J - J_{h,p} = -\mathcal{R}_{h,p}(u_{h,p}, \psi), \tag{4.7}$$

where the adjoint solution $\psi \in W \equiv V + V_{h,p}$ satisfies

$$\overline{\mathcal{R}}'_{h,p}[u, u_{h,p}](w, \psi) = \overline{\mathcal{J}}'_{h,p}[u, u_{h,p}](w), \quad \forall w \in W. \tag{4.8}$$

Here, $\overline{\mathcal{R}}'_{h,p}[u, u_{h,p}] : W \times W \to \mathbb{R}$ and $\overline{\mathcal{J}}'_{h,p}[u, u_{h,p}] : W \to \mathbb{R}$ are the mean-value linearizations defined by

$$\begin{aligned}\overline{\mathcal{R}}'_{h,p}[u, u_{h,p}](w, v) &\equiv \int_0^1 \mathcal{R}'_{h,p}[\theta u + (1-\theta)u_{h,p}](w, v)d\theta \\ \overline{\mathcal{J}}'_{h,p}[u, u_{h,p}](w) &\equiv \int_0^1 \mathcal{J}'_{h,p}[\theta u + (1-\theta)u_{h,p}](w)d\theta,\end{aligned}$$

where $\mathcal{R}'_{h,p}[z](\cdot,\cdot)$ and $\mathcal{J}'_{h,p}[z](\cdot)$ denote the Frechét derivative of $\mathcal{R}_{h,p}(\cdot,\cdot)$ and $\mathcal{J}_{h,p}(\cdot)$ with respect to the first argument evaluated about the state z.

As Eq. (4.8) involves an infinite dimensional space W and also the exact solution u, the adjoint solution ψ is in general not computable. In this work, we approximate ψ by $\psi_{h,p+1}$ in an enriched space $V_{h,p+1} \supset V_{h,p}$, computed from linearization about $u_{h,p}$:

$$\mathcal{R}'_{h,p+1}[u_{h,p}](v, \psi_{h,p+1}) = \mathcal{J}'_{h,p+1}[u_{h,p}](v), \quad \forall v \in V_{h,p+1}.$$

The output error is then estimated by

$$\mathcal{E}_{\text{true}} \approx -\mathcal{R}_{h,p}(u_{h,p}, \psi_{h,p+1}). \tag{4.9}$$

For the purpose of adaptation, a localized error estimate is also defined for each element K by

$$\eta_K \equiv \left|\mathcal{R}_{h,p}(u_{h,p}, \psi_{h,p+1}|_K)\right|, \tag{4.10}$$

where $|_K$ denotes the restriction on the element K. A conservative error estimate for the output of interest is then obtained by the summation of the locally positive error estimate:

$$\mathcal{E} \equiv \sum_{K \in \mathcal{T}_h} \eta_K. \tag{4.11}$$

Note that in Eq. (4.10), the residual is computed about p instead of $p+1$ so that the resulting error estimate is both globally and locally convergent; see [125] for a detailed analysis.

4.4 Output-Based Adaptation

While the DWR method gives a localized output error for each element, the method does not provide enough information for an anisotropic adaptation. Anisotropic information for a simplex element K can be formulated as a metric tensor $\mathcal{M}_K$, often named as the implied metric of K, which is a symmetric positive definite matrix encoding the information of element size and orientation; see, for example, [57, 121]. While a collection of the metric tensors, $\{\mathcal{M}_K\}_{K\in\mathcal{T}_h}$, provides a discontinuous tensor field in the computational domain Ω, a continuous representation $\{\mathcal{M}(\mathbf{x})\}_{\mathbf{x}\in\Omega}$ can also be constructed [24, 80]. Given a mesh $\mathcal{T}_h$, the field $\{\mathcal{M}_K\}_{K\in\mathcal{T}_h}$ is uniquely defined; on the other hand, given a metric tensor field (continuous or not), a family of non-unique discrete meshes can conform to the given field. An example of metric-mesh pair is shown in Figure 4-2. In the work of Loseille and Alauzet [80, 81], this family of metric-conforming meshes is proven to have similar approximation properties with linear polynomials. An extension of their theory to higher-order polynomials is provided by Yano [124], who further proved that the output error for the DG discretization is also a function of the metric field. This lays the foundation for metric-based adaptation algorithms, which strive to decrease the output error (or interpolation error) by manipulating the metric field. In this work, we extend the metric optimization

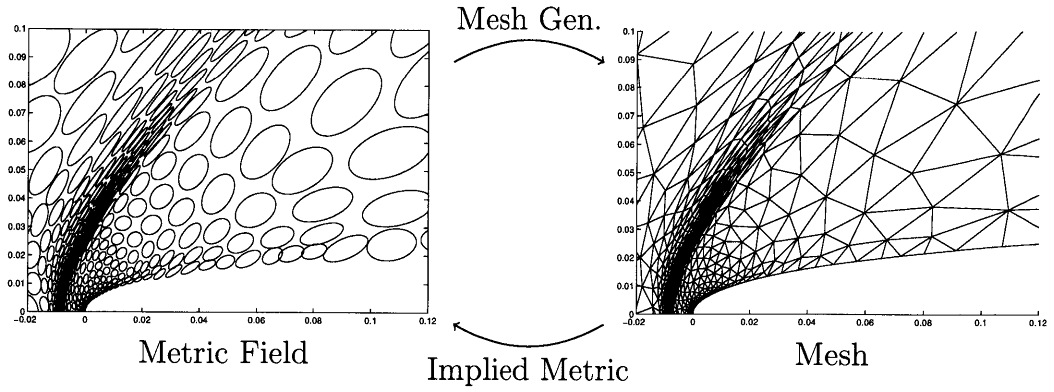


Figure 4-2: Example of metric-mesh pair (Modisette [87])

framework proposed by Yano and Darmofal [127] to handle cut-cell meshes.

In addition, many anisotropic mesh generators take a prescribed metric tensor field as input. In this work, we use the Bidimensional Anisotropic Mesh Generator (BAMG) [21, 59] developed by INRIA to generate all two-dimensional metric-conforming meshes, and the Edge Primitive Insertion and Collapse (EPIC) [85] developed by The Boeing Company for three dimensions.

4.4.1 Mesh Optimization via Error Sampling and Synthesis

The adaptation scheme used in this work is the Mesh Optimization via Error Sampling and Synthesis (MOESS) algorithm developed by Yano and Darmofal [127]. This section briefly reviews the MOESS algorithm, in preparation for presenting its extension to handle cut cells in Section 4.4.2.

The objective of mesh adaptation is to improve the triangulation $\mathcal{T}_h$ for a better output prediction. This can be formulated as an optimization problem, which is to find the optimal triangulation $\mathcal{T}_h^*$:

$$\mathcal{T}_h^* = \arg\inf_{\mathcal{T}_h} \mathcal{E}(\mathcal{T}_h) \quad \text{subject to} \quad \mathcal{C}(\mathcal{T}_h) \leq \text{dof}_{\text{target}}, \tag{4.12}$$

where $\mathcal{E}(\cdot)$ denotes the error functional, and the cost functional $\mathcal{C}(\cdot)$ measures the number of degrees of freedom (DOF) on $\mathcal{T}_h$. As the triangulation $\mathcal{T}_h$ is defined by both the node locations and node connectivity, this continuous-discrete optimization problem is in general intractable. Loseille and Alauzet proposed a continuous relaxation of this optimization problem [79], by appealing to the fact that the metric field, $\boldsymbol{\mathcal{M}} \equiv \{\mathcal{M}(\mathbf{x})\}_{\mathbf{x}\in\Omega}$, controls the approximation properties of a metric-conforming mesh. The relaxed problem reads as: find the optimal $\boldsymbol{\mathcal{M}}^*$ such that

$$\boldsymbol{\mathcal{M}}^* \equiv \arg\inf_{\boldsymbol{\mathcal{M}}} \mathcal{E}(\boldsymbol{\mathcal{M}}) \quad \text{subject to} \quad \mathcal{C}(\boldsymbol{\mathcal{M}}) \leq N. \tag{4.13}$$

The cost functional $\mathcal{C}(\boldsymbol{\mathcal{M}})$ is given by

$$\mathcal{C}(\boldsymbol{\mathcal{M}}) = \int_\Omega c_p \sqrt{\det(\mathcal{M}(\mathbf{x}))} d\mathbf{x}, \tag{4.14}$$

where c_p is a constant for each element dependent on the solution polynomial order. As for the error functional $\mathcal{E}(\boldsymbol{\mathcal{M}})$, a locality assumption for the output error is made in the MOESS algorithm. Under the assumption, each elemental error contribution η_K is a function of the elemental metric tensor: $\eta_K = \eta_K(\mathcal{M}_K)$, and the output functional $\mathcal{E}(\boldsymbol{\mathcal{M}})$ in (4.13) can be expressed as

$$\mathcal{E}(\boldsymbol{\mathcal{M}}) \approx \sum_{K\in\mathcal{T}_h} \eta_K(\mathcal{M}_K). \tag{4.15}$$

The local error function $\eta_K(\mathcal{M}_K)$ is in general not known analytically, and so a surrogate model is constructed.

Local Error Sampling The construction of the surrogate model is achieved by directly monitoring the elemental error change on different local configurations. More specifically, for an element K, let $\{\mathcal{C}_i\}_{i=1}^{n_{\text{config}}}$ denote a set of new configurations, each of which is due to splitting one or multiple edges of K. By convention, $\mathcal{C}_0$ denotes the original configuration. A metric $\mathcal{M}_{\mathcal{C}_i}$ is associated to the configuration $\mathcal{C}_i$ based on the metric tensors of elements on $\mathcal{C}_i$; see Figure 4-3 for an example. On each configuration $\mathcal{C}_i$, a local problem is solved: find the local solution $u_{h,p}^{\mathcal{C}_i} \in V_{h,p}(\mathcal{C}_i)$ such that

$$\mathcal{R}_{h,p}^{\mathcal{C}_i}(u_{h,p}^{\mathcal{C}_i}, v) = 0, \quad \forall v \in V_{h,p}(\mathcal{C}_i), \tag{4.16}$$

where the local semi-linear form $\mathcal{R}_{h,p}^{\mathcal{C}_i}(\cdot,\cdot)$ is from the DG discretization (4.5), and prescribes the boundary fluxes on $\mathcal{C}_i$ by assuming the solution on the neighbor elements does not change. A localized error estimate corresponding to $\mathcal{C}_i$ is then computed:

$$\eta_{\mathcal{C}_i} = \left|\mathcal{R}_{h,p}(u_{h,p}^{\mathcal{C}_i}, \psi_{h,p+1}|_K)\right|. \tag{4.17}$$

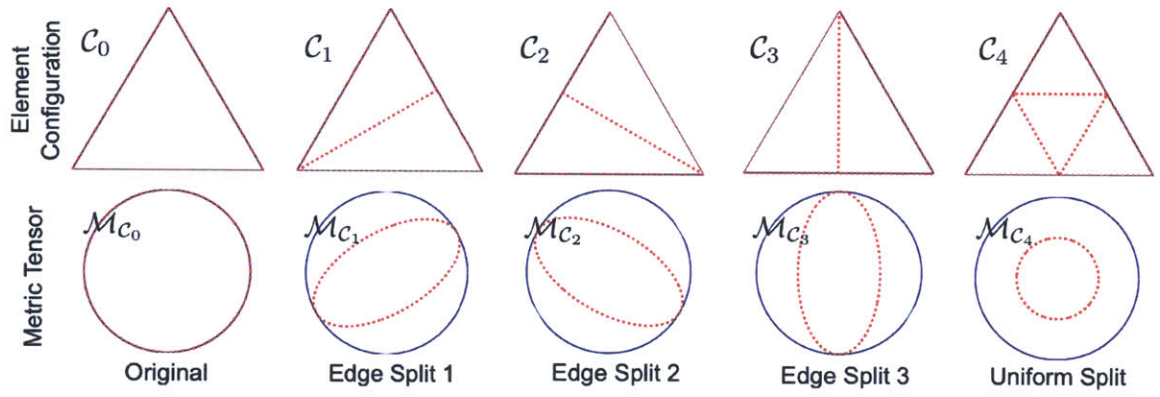


Figure 4-3: Example configurations together with the associated metric tensors (Yano [124])

Local Error Model Synthesis After collecting the set of metric-error pairs, $\{\mathcal{M}_{\mathcal{C}_i}, \eta_{\mathcal{C}_i}\}_{i=1}^{n_{\text{config}}}$, a continuous local error model $\eta_K(\cdot) : Sym_d^+ \to \mathbb{R}^+$ is then synthesized. More specifically, the change in the metric tensor from $\mathcal{C}_0$ to a new configuration $\mathcal{C}_i$ is measured based on the affine invariant framework [94]:

$$\mathcal{S}_{\mathcal{C}_i} \equiv \log\left(\mathcal{M}_{\mathcal{C}_0}^{-1/2}\mathcal{M}_{\mathcal{C}_i}\mathcal{M}_{\mathcal{C}_0}^{-1/2}\right), \tag{4.18}$$

and the change in error is defined as

$$f_{\mathcal{C}_i} \equiv \log\left(\eta_{\mathcal{C}_i}/\eta_{\mathcal{C}_0}\right). \tag{4.19}$$

The proposed error model in the MOESS algorithm has a form of

$$f_K(\mathcal{S}_K) = \mathrm{tr}(\mathcal{R}_K \mathcal{S}_K),$$

where the matrix $\mathcal{R}_k$ is synthesized from the pairs $\{\mathcal{S}_{\mathcal{C}_i}, f_{\mathcal{C}_i}\}_{i=1}^{n_{\mathrm{config}}}$, and represents the local error sensitivity with respect to the local element shape and size as argued in [124]. The local error model is then in terms of $\mathcal{S}$:

$$\eta_K(\mathcal{S}_K) = \eta_{\mathcal{C}_0} \exp(\mathrm{tr}(\mathcal{R}_K \mathcal{S}_K)). \tag{4.20}$$

Model Optimization Once the local error model is constructed, the optimization problem (4.13) is then solved using a gradient-based method. In particular, the metric field $\boldsymbol{\mathcal{M}}$ is represented using the vertex values $\{\mathcal{M}_v\}_{v\in\mathcal{V}(\mathcal{T}_h)}$, and the element metric change $\mathcal{S}_K$ is assigned based on the vertex value $\mathcal{S}_v$ via an arithmetic average:

$$\mathcal{S}_K = \overline{\{\mathcal{S}_v\}_{v\in\mathcal{V}(K)}} \equiv \frac{1}{|\mathcal{V}(K)|} \sum_{v\in\mathcal{V}(K)} S_v,$$

where $\mathcal{V}(K)$ denotes the vertices of K. Then the objective function in (4.13) (or equivalently (4.15)) becomes

$$\mathcal{E}(\{S_v\}_{v\in\mathcal{V}(\mathcal{T}_h)}) \approx \sum_{K\in\mathcal{T}_h} \eta_K\left(\overline{\{S_v\}_{v\in\mathcal{V}(K)}}\right),$$

where the design variables are $\{S_v\}_{v\in\mathcal{V}(\mathcal{T}_h)}$. The gradient is computed by

$$\frac{\partial\mathcal{E}}{\partial S_v} = \sum_{K\in\omega(v)} \left[\eta_K\left(\overline{\{S_v\}_{v\in\mathcal{V}(K)}}\right) \frac{1}{|\mathcal{V}(K)|}\mathcal{R}_K\right], \tag{4.21}$$

where $\omega(v)$ denotes the set of elements adjoining the vertex v.

4.4.2 Extension to Cut Cells

In this section, we extend the MOESS algorithm to handle cut elements with arbitrary shapes. To achieve this, the goal of adaptation for a cut-cell mesh is defined to improve the background triangulation $\mathcal{T}_{h,b}$. The optimization problem (4.12) becomes to find the optimal $\mathcal{T}_{h,b}^*$:

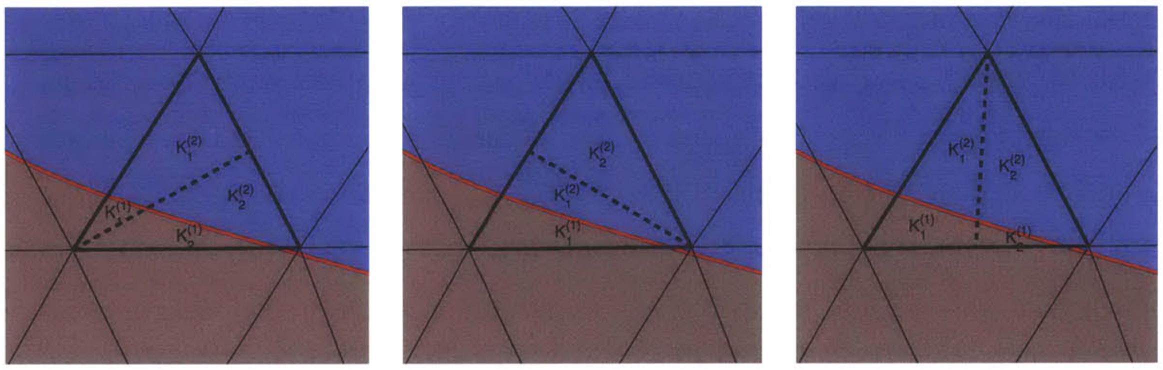
$$\mathcal{T}_{h,b}^* = \arg \inf_{\mathcal{T}_{h,b}} \mathcal{E}(\mathcal{T}_{h,b}) \quad \text{subject to} \quad \mathcal{C}(\mathcal{T}_{h,b}) \leq \text{dof}_{\text{target}}, \tag{4.22}$$

where $\mathcal{E}(\mathcal{T}_{h,b})$ and $\mathcal{C}(\mathcal{T}_{h,b})$ measure the error and DOF evaluated on the cut mesh $\mathcal{T}_h$, which is generated from intersecting $\mathcal{T}_{h,b}$ and the embedded geometry. Following the same approach as for problem (4.12), we consider a continuous relaxation of (4.22), by assuming the metric field, $\boldsymbol{\mathcal{M}} \equiv \{\mathcal{M}(\mathbf{x})\}_{\mathbf{x}\in\Omega}$, controls the discretization error of a metric-conforming mesh that may not be geometry-conforming. The relaxed problem has the same formulation as (4.13), but the error functional $\mathcal{E}(\boldsymbol{\mathcal{M}})$ is approximated by

$$\mathcal{E}(\boldsymbol{\mathcal{M}}) \approx \sum_{K_b \in \mathcal{T}_{h,b}} \eta_{K_b}(\mathcal{M}_{K_b}),$$

where η_{K_b} is the sum of errors on all the cut elements generated from K_b, and is assumed to be a function of $\mathcal{M}_{K_b}$. Note η_{K_b} is set to zero when K_b is outside the computational domain.

Same as in Section 4.4.2, a local error model $\eta_{K_b}(\mathcal{S}_{K_b})$ is constructed in the form of (4.20) for each background element K_b. The matrix $\mathcal{R}_{K_b}$, representing the local error sensitivity, is again constructed through local error re-computation on different configurations of K_b. On each configuration $\mathcal{C}_i$, we first construct the cut elements by intersecting $\mathcal{C}_i$ with the embedded geometry. Figure 4-4 shows an example of different configurations due to edge splits of a background element. We then perform local solve (4.16) on these cut elements, and evaluate the corresponding error estimate $\eta_{\mathcal{C}_i}$ from (4.17). The associated metric $\mathcal{M}_{\mathcal{C}_i}$ is computed based on the metric tensors of the split background elements on $\mathcal{C}_i$. The changes in the error and the metric, $f_{\mathcal{C}_i}$ and $\mathcal{S}_{\mathcal{C}_i}$, are computed from (4.19) and (4.18), respectively. The error model $\eta_{K_b}(\mathcal{S}_{K_b})$ is then synthesized from the collected samples $\{\mathcal{S}_{\mathcal{C}_i}, f_{\mathcal{C}_i}\}_{i=1}^{n_{\text{config}}}$, and the same optimization algorithm as before is applied to obtain a description of vertex-based metric change, $\{\mathcal{S}_v\}_{v \in \mathcal{V}(\mathcal{T}_{h,b})}$.



(a) Configuration $\mathcal{C}_1$ (b) Configuration $\mathcal{C}_2$ (c) Configuration $\mathcal{C}_3$

Figure 4-4: Example of different configurations due to edge splits; configurations $\mathcal{C}_1$ and $\mathcal{C}_3$ have four cut elements, and $\mathcal{C}_2$ has three

Note for a background element K_b that intersects with the embedded geometry, the error-metric relationship can be "noisy". The reason is as following: while the error η_{K_b} is evaluated from the cut elements generated from K_b, the metric $\mathcal{M}_{K_b}$ may not well represent the shapes of these cut elements. However, on many of the intersecting background elements, the surrogate error model we construct still has a valid error sensitivity information, $\mathcal{R}_{K_b}$. This is in particular true when the cut cell occupies a large fraction of the volume of its parent background element. For instance in Figure 4-4, if the underlying PDE solution has a thin anisotropic layer along the embedded geometry (in the sub-domain above the geometry), the configurations $\mathcal{C}_1$ and $\mathcal{C}_2$ will likely have a lower error than $\mathcal{C}_3$, and the surrogate error model with this information will drive the optimization algorithm toward a better background mesh. In addition, the optimization is based on the vertex-based metric description, where the gradient for a vertex v is computed from an error-weighted average of the error sensitivities in the surrounding elements as in (4.21). This vertex-based gradient will therefore lead to an improved mesh, as long as not all the surrounding elements have a poor-quality sensitivity information, $\mathcal{R}_{K_b}$.

On a cut-cell mesh for an embedded-boundary problem, the null elements outside the computational domain Ω do not have an associated error or error model. The vertices surrounded by such elements, named as null vertices, are thus not considered in the metric optimization problem (4.13). However, we still need to define metrics on them, $\{\mathcal{M}_v\}_{v \in \mathcal{V}^{\text{null}}}$, for the purpose of generating an adapted background mesh. These metrics can have an impact on the elements generated near the embedded boundary, and so it is desirable to have $\{\mathcal{M}_v\}_{v \in \mathcal{V}^{\text{null}}}$ similar to the optimized metrics near the boundary inside the computational domain. On the other hand, filling the null region with a large number of elements may slow down the mesh generation process. Therefore, the metrics $\{\mathcal{M}_v\}_{v \in \mathcal{V}^{\text{null}}}$ are assigned as follows.

We first assign a layer number $\mathcal{L}$ for each vertex on $\mathcal{T}_{h,b}$. Any vertex v on a background element that has any fraction inside Ω is assigned $\mathcal{L}(v) = 0$; then a vertex v with $\mathcal{L}(v) = n > 0$ means there is a path consisting of n edges on $\mathcal{T}_{h,b}$ from a layer-0 vertex to v. An example of layer numbers is shown in Figure 4-5, where the computational domain is on top of the embedded geometry. The requested metrics on layer-0 vertices are from the MOESS algorithm. For a vertex with $\mathcal{L} = n > 0$, its requested metric is set to be the barycentric average of the requested metrics on its neighbor vertices with $\mathcal{L} = n - 1$. Further, we apply a volume growth rate of 1.15 between the requested metrics for vertices on levels $n - 1$ and n.

4.4.3 Results

In this section, we present two numerical examples for the MOESS algorithm for cut cells, and compare with the results on boundary-conforming meshes. In the first example, we solve an advection-diffusion equation on a computational domain with embedded interfaces of different shapes. The second example presents a subsonic,

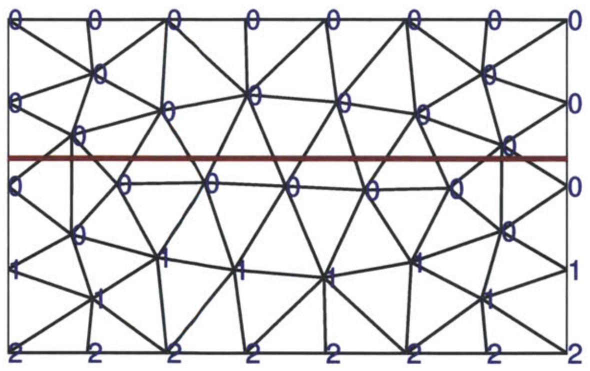


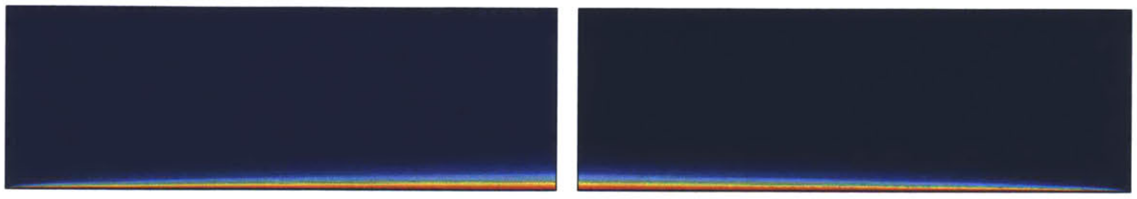
Figure 4-5: Example of vertex layer numbers; computational domain is on top of the embedded geometry denoted by the red line

turbulent flow over a RAE2822 airfoil, which is treated as an embedded boundary in the background mesh. Note the governing equation for this case is the Reynolds-Averaged Navier-Stokes equations, with the Spalart-Allmaras turbulence model [115]; see Appendix E.2.2 for detailed description.

Example 1: Advection-Diffusion Equation, Arbitrary Embedded Interfaces

This example is taken from [127], where we solve an advection-diffusion equation with a Peclet number of 10^{-3} on a rectangular domain. The solution has a boundary layer along the bottom wall, shown in Figure 4-6(a). The output of interest is the heat transfer across the bottom wall, and the associated dual solution is shown in Figure 4-6(b). We first solve the problem with the MOESS algorithm on boundary-conforming meshes. The DG solution polynomials have an order of $p = 2$. Figure 4-7 shows the initial mesh, and Figure 4-8 shows the adapted mesh obtained from the MOESS algorithm with a target DOF of 1800.

We then introduce interfaces of different shapes into the computation domain. Note the PDE parameters on the two sides of the introduced interfaces remain the same, i.e. the interfaces do not alter the PDE solution. On the other hand, arbitrarily-shaped elements are created along the interfaces, and the MOESS algorithm for cut cells is tested. The initial mesh is the same as in Figure 4-7, and the target DOF is still 1800. Figure 4-9 shows the adapted meshes for each introduced interface (shown in red line). It is seen that these meshes are similar to each other, and also similar to the adapted mesh without interface in Figure 4-8. Figure 4-10 shows the adaptation history, where all the cases have the same error convergence. Note the cut-cell cases have slightly higher DOF because each background element on the interface is cut into multiple cut elements.



(a) Primal solution (b) Dual solution

Figure 4-6: Solutions to the advection-diffusion boundary-layer problem

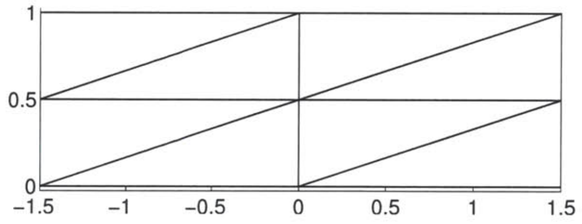
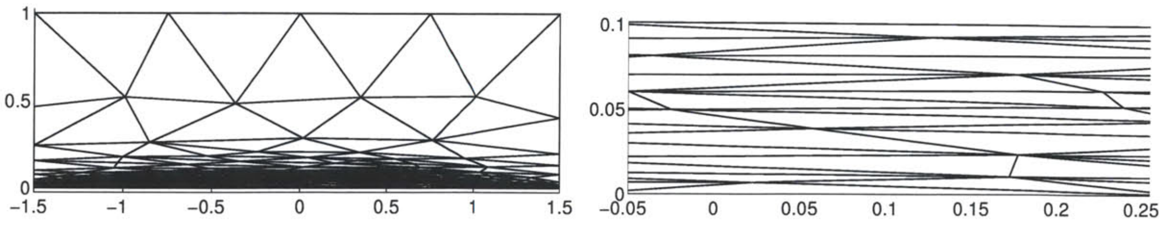
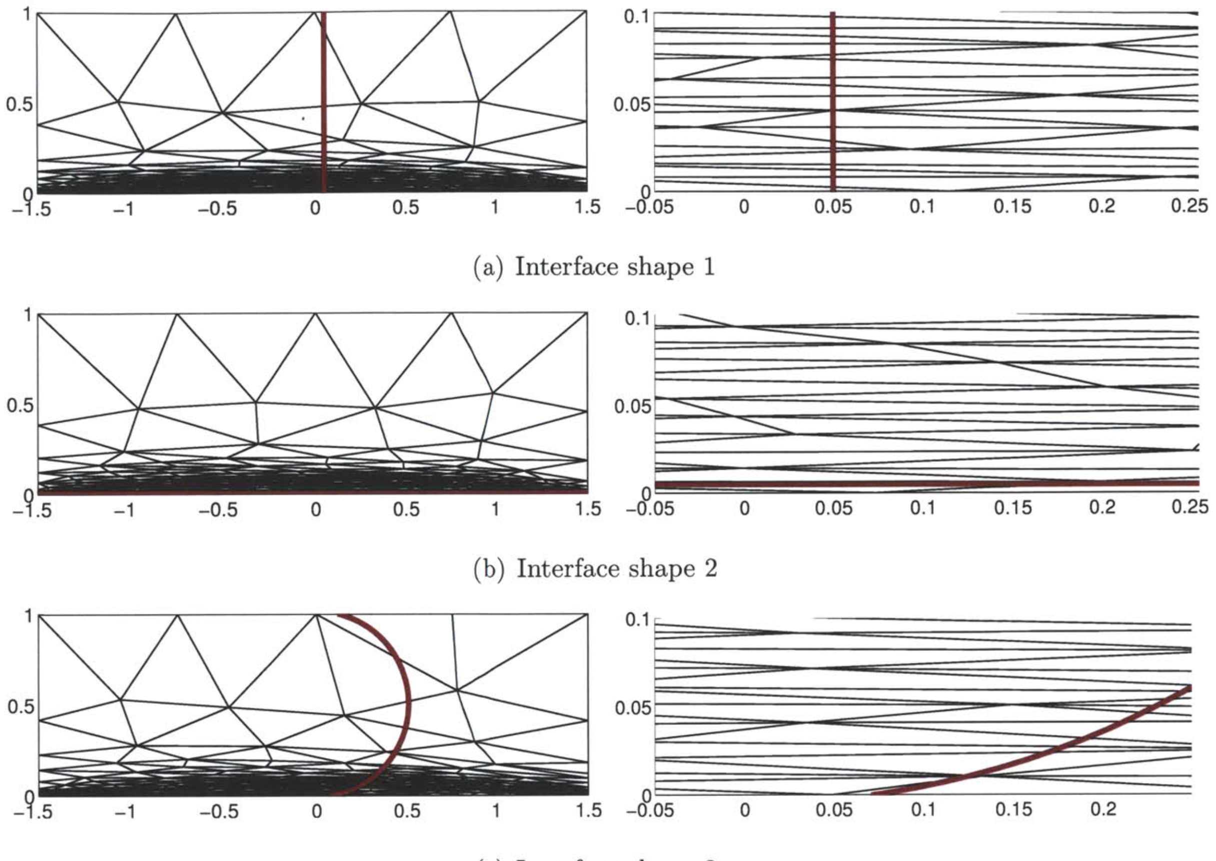


Figure 4-7: Initial mesh for the advection-diffusion boundary-layer problem



(a) Adapted mesh (b) Adapted mesh, zoom-in

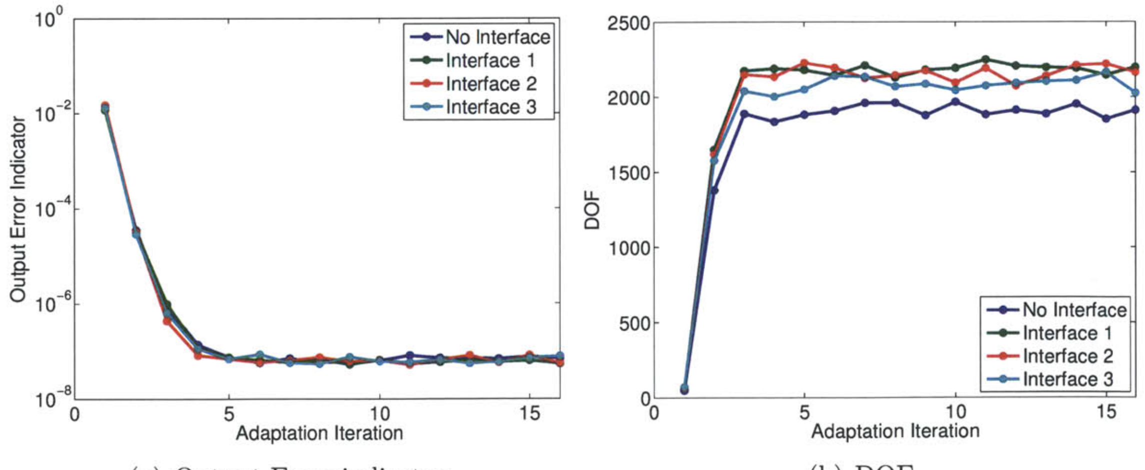
Figure 4-8: Adapted mesh for the advection-diffusion boundary-layer problem without interface introduced


(a) Interface shape 1

(b) Interface shape 2

(c) Interface shape 3

Figure 4-9: Adapted cut-cell meshes for the advection-diffusion boundary-layer problem with different interface shapes



(a) Output Error indicator (b) DOF

Figure 4-10: Adaptation history for the advection-diffusion boundary-layer problem

Example 2: RANS, RAE2822 Airfoil

In this example, we consider a subsonic, turbulent flow over a RAE2822 airfoil. The freestream Mach number is $M_\infty = 0.3$, the Reynolds number is $Re_c = 6.5 \times 10^6$, and the angle of attack is $\alpha = 2.31°$. The Mach number distribution for this case is shown in Figure 4-11. The MOESS algorithm is applied on both boundary-conforming and cut-cell meshes. The DG solution has a polynomial degree of $p = 2$, and the target DOF is 20k. The initial boundary-conforming mesh is an isotropic mesh, as shown in Figure 4-12(a). This mesh is then used to make the initial cut-cell mesh, where we move the boundary nodes on the airfoil into the null region by a small distance, and triangulate the region using only these nodes. Figure 4-12(b) shows the initial cut-cell mesh, which has essentially the same topology as the initial boundary-conforming one.

Figure 4-13 shows the adaptation history for 20 adaptation iterations. In the early iterations where the elements in the boundary layer gradually transition to anisotropic shapes, the cut-cell case has a higher error and takes a couple more adaptation iterations for this transition. This is due to the fact that the local error model for some cut cells does not result in a correct error to metric sensitivity. The adapted meshes at iteration 8 are shown in Figure 4-14, where the boundary-conforming mesh has a more appropriate anisotropy in the boundary layer. As the adaptation progresses, both cut-cell and boundary-conforming cases reach the same error level, and both capture the boundary layer with highly anisotropic elements, shown in Figure 4-15. Note for adaptation at a higher DOF, we can start from the mesh adapted at a lower DOF (if available), which usually already possesses an appropriate anisotropy. Starting from such a mesh, the impact of the poorer-quality error model on cut elements is decreased, and the boundary-conforming and cut-cell adaptation histories are even more similar.

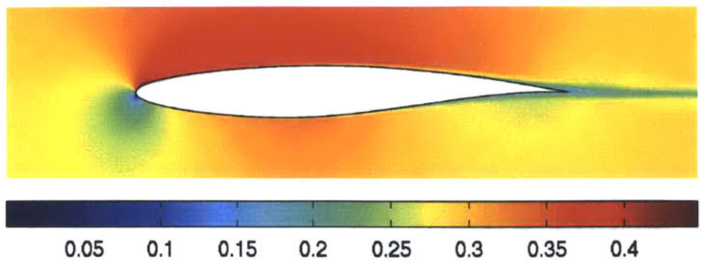
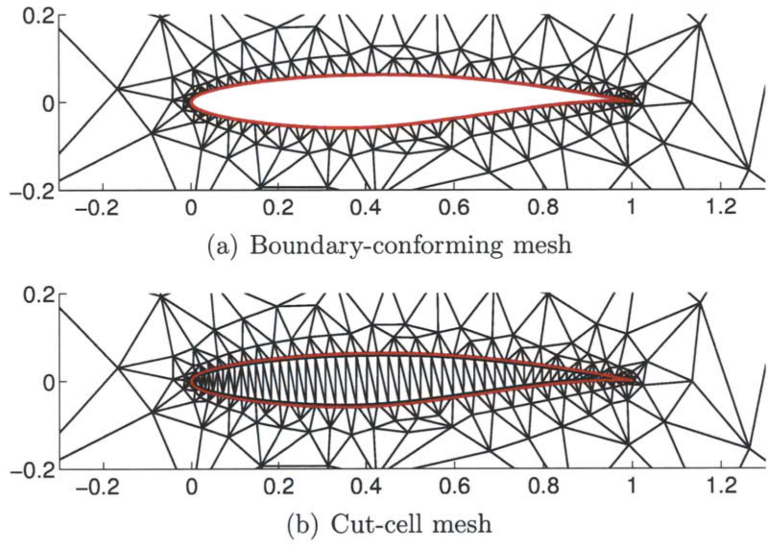


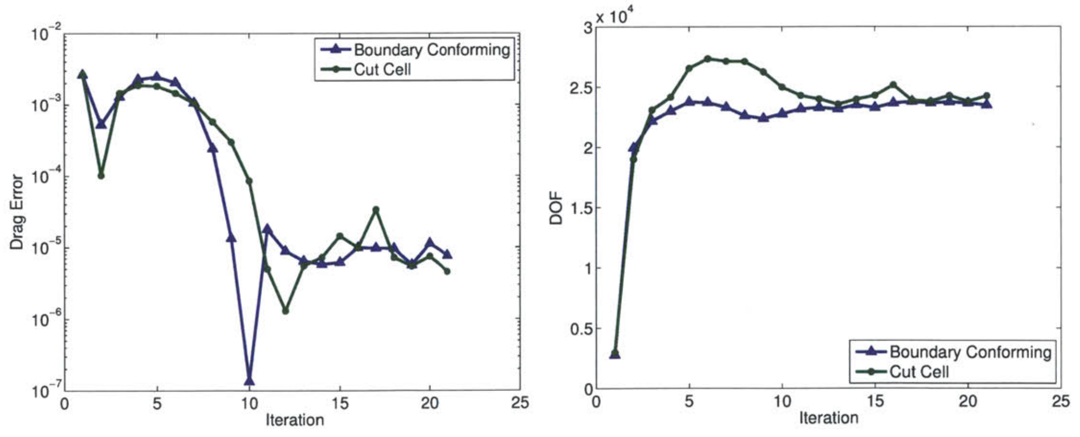
Figure 4-11: Mach number distribution for the RAE2822 subsonic RANS-SA problem



(a) Boundary-conforming mesh

(b) Cut-cell mesh

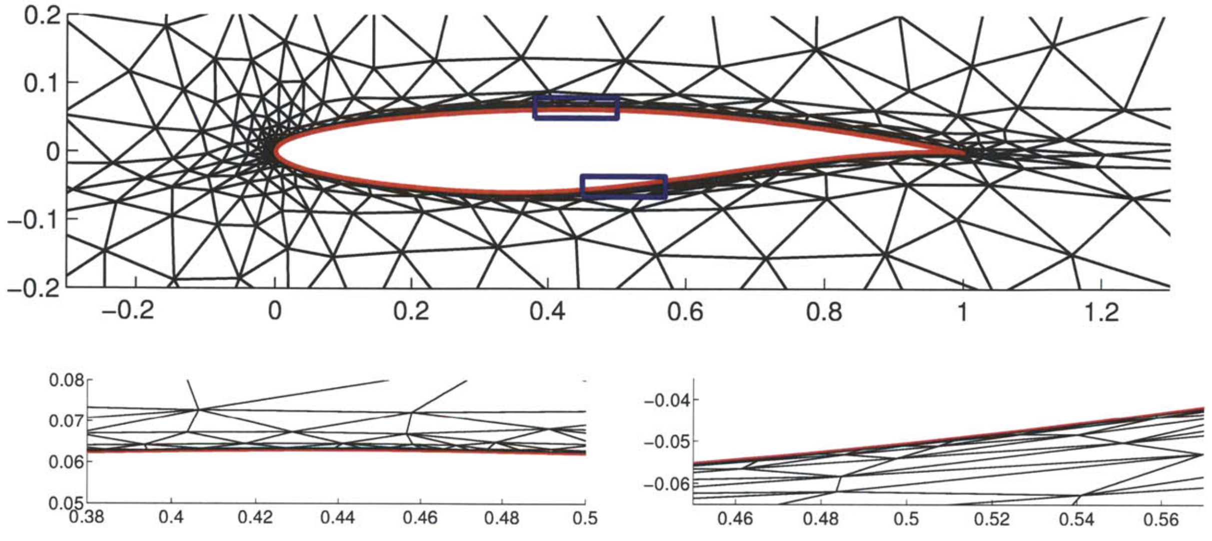
Figure 4-12: Initial meshes for the RAE2822 subsonic RANS-SA problem



(a) Drag Error

(b) DOF

Figure 4-13: Adaptation history for the RAE2822 subsonic RANS-SA problem


(a) Boundary-conforming mesh

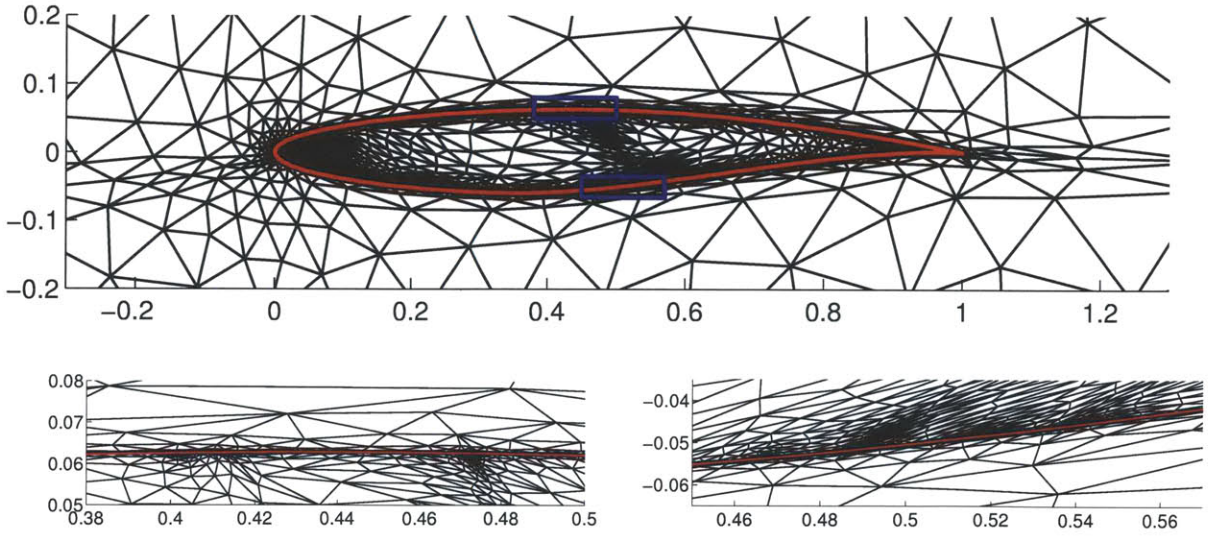
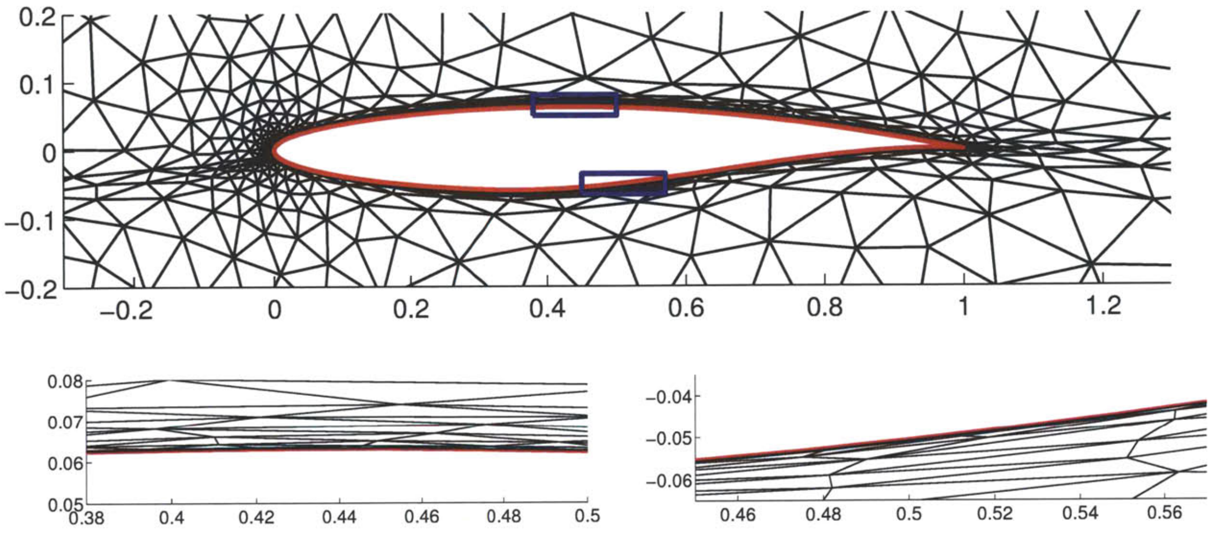
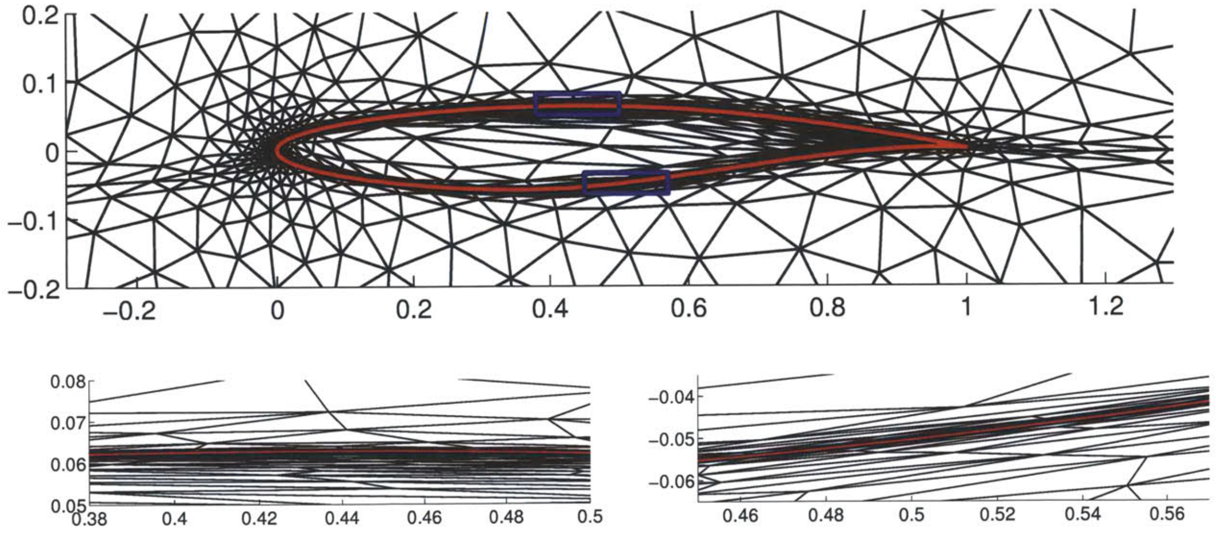

(b) Cut-cell mesh

Figure 4-14: Adapted meshes, iteration 8, with zoom-in for the blue-box regions; RAE2822, subsonic RANS-SA



(a) Boundary-conforming mesh



(b) Cut-cell mesh

Figure 4-15: Adapted meshes, iteration 20, with zoom-in for the blue-box regions; RAE2822, subsonic RANS-SA

Chapter 5

Aerodynamics Problems in Three Dimensions

This chapter applies our solution framework to three-dimensional aerodynamic flows governed by Euler and compressible Navier-Stokes equations; see Appendix E for a description of these equations. In particular, Section 5.1 shows the robustness improvement for nonlinear solvers using the proposed cut-cell quadrature rule. Note in this work, the nonlinear solver applies a pseudo-time stepping to march the solution to a steady state using a backward Euler integrator; see Appendix F.3 for details. In Section 5.2, we demonstrate the robustness and automation of our framework through a range of aerodynamics problems, including subsonic through supersonic regimes. Note the adaptation scheme in this chapter is based on a fixed-fraction strategy [128], with anisotropy detection based on higher-order derivatives of the Mach number [50]. The MOESS adaptation algorithm has not been implemented for cut-cell meshes in three dimensions. Generation of the (three-dimensional) background meshes is carried out using the Edge Primitive Insertion and Collapse (EPIC) [85] developed by The Boeing Company.

5.1 Impact of Quadrature Rules

As quadrature rules are typically designed to be exact for certain polynomials, quadrature error is introduced in the residual evaluations for a DG discretization of Navier-Stokes equations, for which the fluxes present non-polynomial functions (of the conservative variables). Such error can result in poor nonlinear solver convergence, especially when higher-order polynomial approximation is employed. This is demonstrated in this section through an example of a subsonic, inviscid flow, where the geometry is a body of revolution defined by

$$r = 0.3x(1-x), \quad y = r\cos(\theta), \quad z = r\sin(\theta), \quad 0 \le x \le 1, \quad 0 \le \theta \le \pi, \tag{5.1}$$

and shown in Figure 5-1. This geometry is also considered in Fidkowski [49], and is named "football" in this chapter. The flow has a freestream Mach number of $M_\infty = 0.3$, an angle of attack of $\alpha = 0$, and an sideslip angle of $\beta = 0$. Figure 5-2 shows the Mach number distribution for this flow.

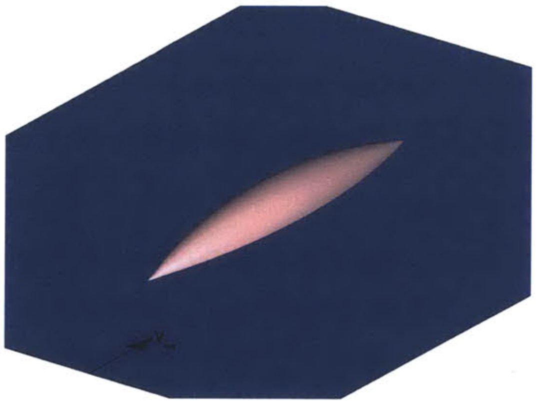


Figure 5-1: "Football" geometry produced by revolving a quadratic curve

To examine the robustness of the nonlinear solver, we consider 25 similar background meshes obtained from the adaptation process. Adaptation results will be discussed in Section 5.2. On each mesh, we attempt to solve the flow using polynomial degrees of $p = 0$ through $p = 3$. At $p = 0$, we start the time-marching from a uniform flow of freestream conditions; and at higher p, we start from the converged

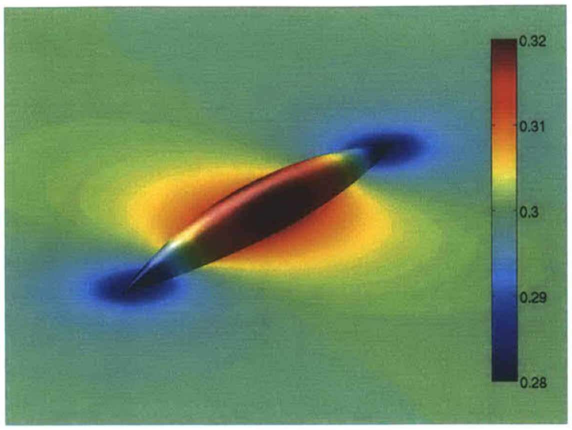
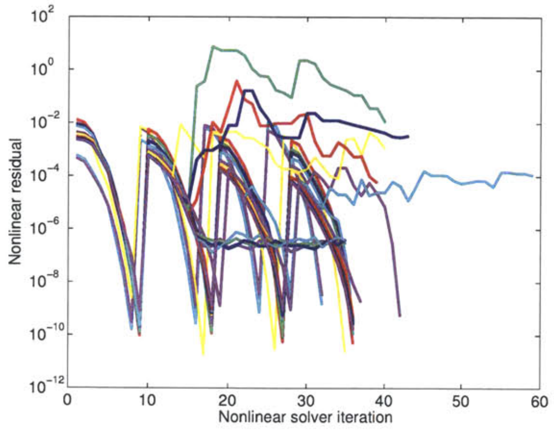


Figure 5-2: Mach number distribution on the "football" geometry, $M_\infty = 0.3$

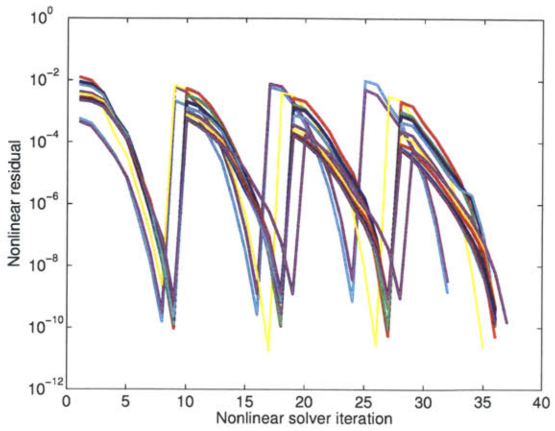
solution at $p-1$.

Two sets of quadrature rules for the cut elements are considered. Both quadrature rules have the same algebraic degree, six for elements and nine for faces. The first consists of a set of uniformly spaced points inside each cut element (and face), with quadrature weights computed from Eq. (3.6). The number of quadrature points is about $2n_b$, where n_b is the dimension of the polynomial space for which the rule integrates exactly. Figure 5-3(a) shows the nonlinear convergence histories on the 25 meshes, where the solver cannot converge on about a third of the meshes even for the $p = 1$ discretization. The quadrature quality measure Q, defined in Eq. (3.7), is shown in red in Figure 5-4. In the figure, each dot represents the quality on each cut element on one of the meshes where the solver did not converge. Denote this mesh by $\mathcal{T}_h^{\text{fail}}$. As shown in the figure, several cut elements have a very large value of Q, resulting in a poor quadrature quality. Similar distribution of Q is also observed on other meshes with convergence difficulty.

The second quadrature rule considered is the cut-cell quadrature rule proposed in Section 3.6. Although this rule has the same algebraic degree as the first rule, the nonlinear solver converges on all the meshes for the DG polynomial degrees of $p = 0$ through $p = 3$ as shown in Figure 5-3(b). Figure 5-4 shows the quality measure Q for each cut element on the mesh $\mathcal{T}_h^{\text{fail}}$, and Table 5.1 lists the number of quadrature points. With at most $1.5n_b$ points, every cut element achieves a quadrature quality measure of $Q < Q^{\text{threshold}} = 5$.



(a) Quadrature rule with uniform points, $n_q \approx 2n_b$



(b) Proposed quadrature rule

Figure 5-3: Nonlinear convergence history on 25 meshes for discretization degrees of $p = 0$ through $p = 3$; each line represents the history on one mesh

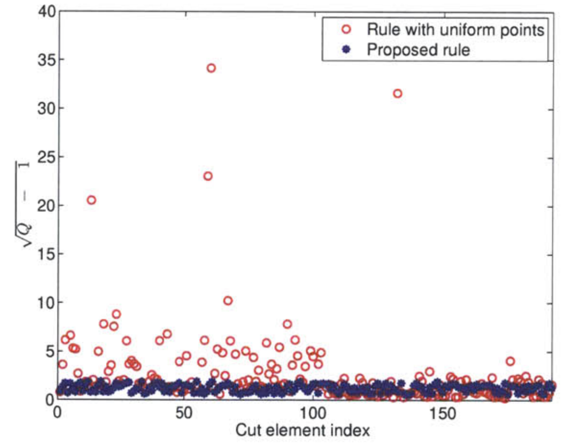


Figure 5-4: Quadrature quality Q for cut elements

n_q	Fraction of cut elements
$n_b \leq n_q \leq 1.25n_b$	60%
$1.25n_b < n_q \leq 1.5n_b$	40%
$n_q > 1.5n_b$	0%

Table 5.1: Number of quadrature points for the proposed rule

5.2 Robustness and Accuracy of the Solution Strategy

The main objective of this section is to demonstrate the robustness of the developed solution strategy through simulations of three-dimensional aerodynamic flows. For each presented case, the cut-cell method is applied on 100 to 200 adapted meshes, ranging from a very coarse mesh with the geometry inside almost one background element, to an adapted mesh with flow features resolved for an accurate output prediction. No human intervention is involved in the process from the initial to the final mesh, including the cut-cell intersection procedure and flow solves using DG polynomial degrees of $p = 1$ and $p = 2$.

5.2.1 "Football", Inviscid, Subsonic

The first case to demonstrate the robustness of the solution strategy is the inviscid subsonic flow over the "football" geometry presented in Section 5.1. The output adaptation is performed upon drag. Solution singularities are present at the leading and trailing tips of the football.

A fixed-DOF adaptation algorithm [128] is applied at several degrees of freedom (DOF) for the DG polynomial degrees of $p = 1$ and $p = 2$. At each DOF, 20 adapted meshes are generated and solved. For the total of 200 meshes, no human interaction is involved from the initial to the final mesh. Figure 5-5 shows the adaptation history with each dot representing one adapted mesh. The true drag value for this flow is nearly zero, but not exactly due to the finite proximity of the domain boundaries. Hence, the C_D plotted in the figure also represents the error in C_D. Note the reference area for computing C_D is the frontal cross-section area. Despite the presence of solution singularities, the optimal output error convergence of h^{2p+1} is observed for both finite element degrees, and the $p = 2$ discretization is superior to $p = 1$ at all considered DOFs and error levels. This is due to strong mesh gradings around the leading and trailing tips achieved on the adapted meshes. Figure 5-6 shows the initial and the adapted meshes on the symmetry plane zoomed at the geometry.

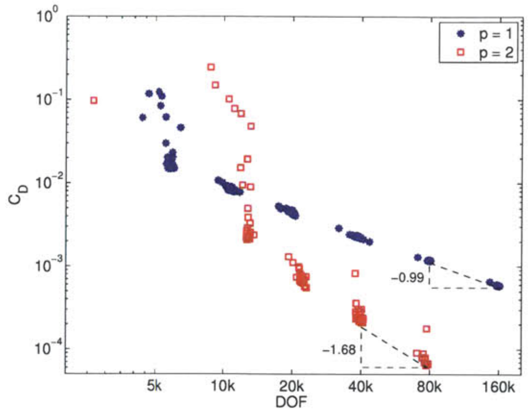
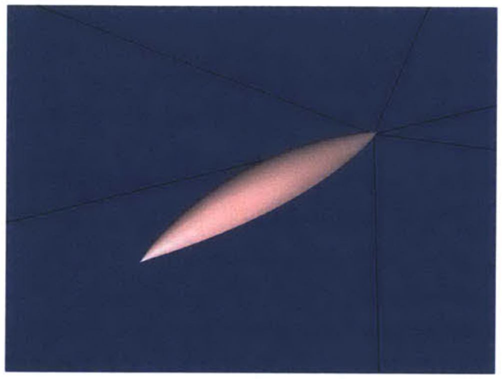
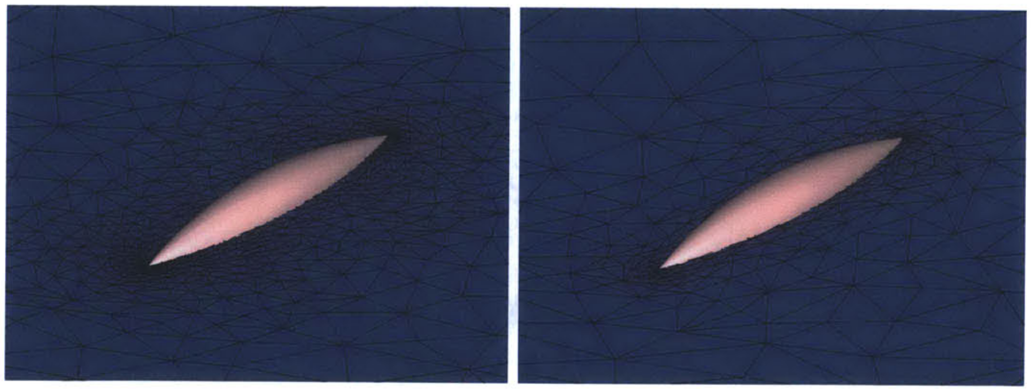


Figure 5-5: Drag adaptation history for the "football" geometry, inviscid, $M_\infty = 0.3$



(a) Initial Mesh



(b) Adapted mesh, $p = 1$, $DOF = 80k$ (c) Adapted mesh, $p = 2$, $DOF = 80k$

Figure 5-6: Initial and adapted meshes on the symmetry plane for the "football" geometry, inviscid, $M_\infty = 0.3$

5.2.2 ONERA M6 Wing, Inviscid, Transonic

This case presents an inviscid transonic flow over the ONERA M6 wing with $M_\infty = 0.8395$, $\alpha = 3.06°$, and $\beta = 0$. The adaptation is based on drag. The Mach number distribution is shown in Figure 5-7, which is obtained from a cut-cell mesh of $DOF = 800k$ using $p = 1$ discretization. Appendix D describes the visualization of DG solutions on cut-cell meshes.

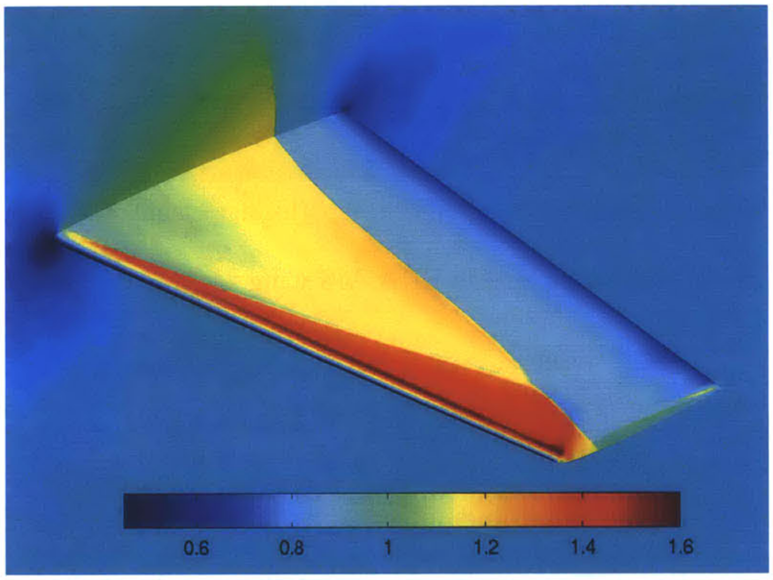
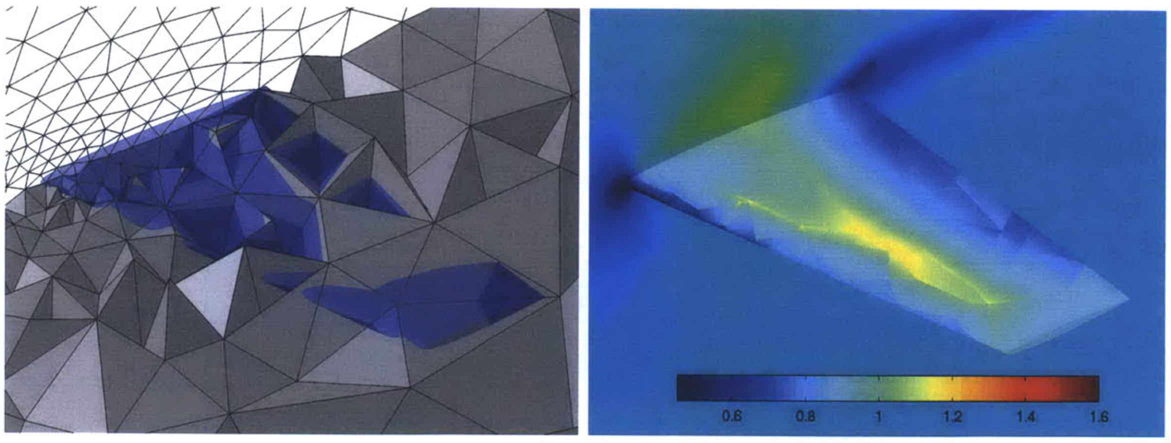


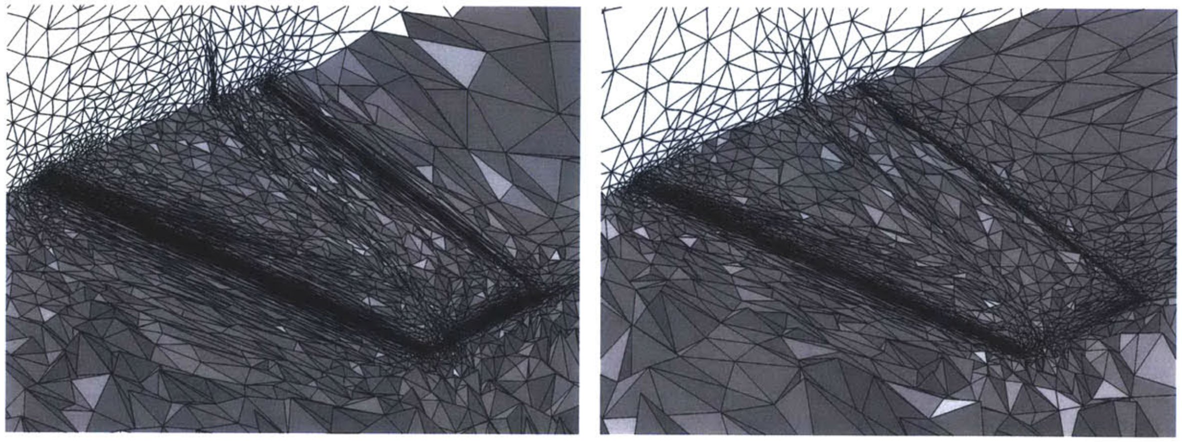
Figure 5-7: Mach number distribution for ONERA M6 wing, $M_\infty = 0.8395$, $\alpha = 3.06°$; solution on the adapted mesh, $p = 1$, $DOF = 800k$

For this case, the same fixed-DOF algorithm is applied. At each DOF, 15 meshes are generated and solved for each p. The initial mesh is shown in Figure 5-8 together with the Mach number solution using the $p = 1$ discretization on this mesh, where the shock waves are not at all resolved. From the same initial mesh, the adaptation using $p = 1$ and $p = 2$ lead to the adapted meshes in Figure 5-9, where all background elements that intersect with the upper surface of the wing are shown. Again, the process from the initial to the adapted meshes is fully-automated. On these adapted meshes, the adaptation employs anisotropic elements along the leading and trailing edges and along the wing tip, and has additional refinement for resolving the shock waves. Note the anisotropic elements on the wing surface pose no issues to the cut-cell intersection algorithm.



(a) Initial mesh (b) Mach number distribution, $p = 1$

Figure 5-8: Initial mesh for ONERA M6 wing, $M_\infty = 0.8395$, $\alpha = 3.06°$

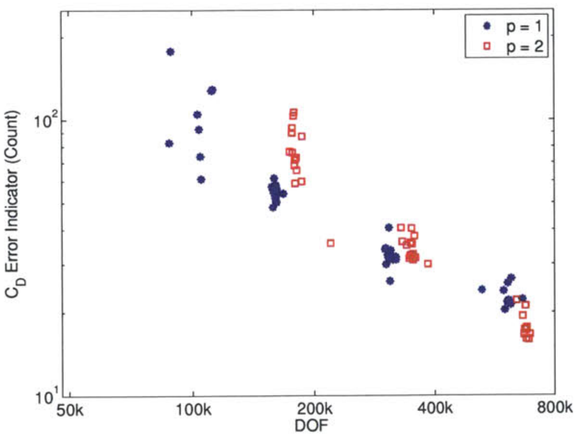


(a) $p = 1$, $DOF = 800k$ (b) $p = 2$, $DOF = 800k$

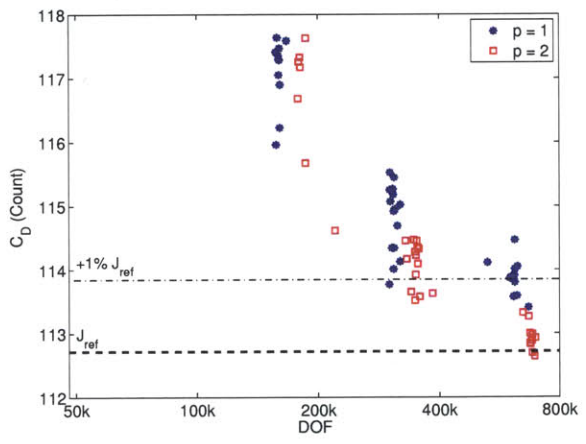
Figure 5-9: Adapted meshes for ONERA M6 wing, $M_\infty = 0.8395$, $\alpha = 3.06°$

Figure 5-10 shows the adaptation history for the drag coefficient C_D, where the planform area is used for computing C_D, and the reference value is obtained using a $p = 2$ discretization with $DOF = 2.5M$. Note this reference value may still have a considerable error compared to the true C_D value for this case. The output error indicator is computed from (4.11), which is the agglomeration of the absolute value of the elemental error estimate, and can overestimate the true output error. For this case where the shock waves are the only dominant features for the output prediction, the benefit of a higher-order discretization is not obvious, especially at lower DOFs. This is consistent with the adaptation results for inviscid transonic flows in two dimensions [126]. However, at a DOF of about $800k$, the $p = 2$ discretization still appears more accurate than the $p = 1$ counterpart, as all the adapted meshes for $p = 2$ predict a drag value within 1% of the reference value.

Note for this case, the $p = 2$ results may be more accurate with the MOESS adaptation. In particular, the requested element anisotropy using the $p + 1$ derivative of the Mach number may have a very poor quality on the trailing edge, where the solution presents an edge singularity and may not have a well-defined third derivative. This is in fact reflected in Figure 5-9(b), where the elements on the trailing edge may not have an optimal anisotropy. With a proper anisotropy detection, fewer DOFs would be employed on the trailing edge, allowing more DOFs to resolve the shock waves.



(a) C_D error indicator



(b) C_D

Figure 5-10: Adaptation history for ONERA M6 wing, $M_\infty = 0.8395$, $\alpha = 3.06°$; reference value is obtained at $p = 2$, $DOF = 2.5M$

5.2.3 "Football", Inviscid, Supersonic

This case has an inviscid supersonic flow over the "football" geometry with $M_\infty = 1.8$, $\alpha = 0$, and $\beta = 0$. The output of interest is the pressure perturbation on the outflow boundary: $\int_{\text{out}}((P - P_\infty)/P_\infty)^2 dS$. The distribution of the pressure perturbation is shown in Figure 5-11.

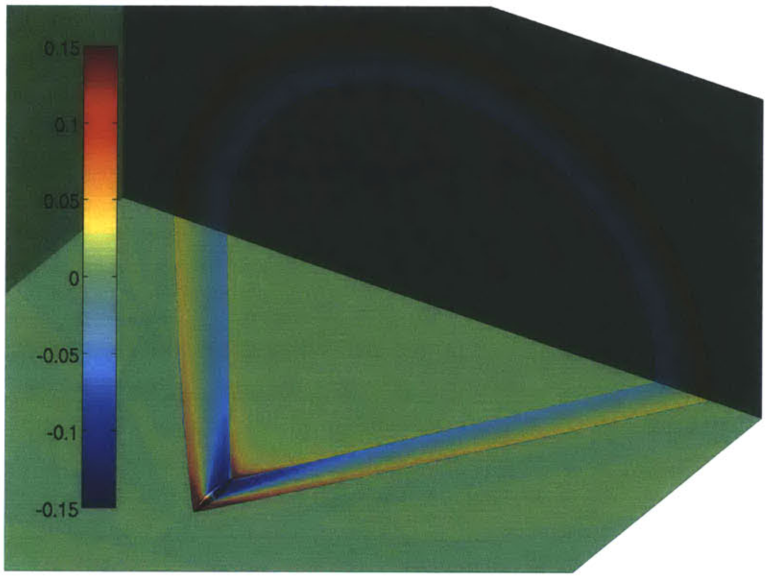
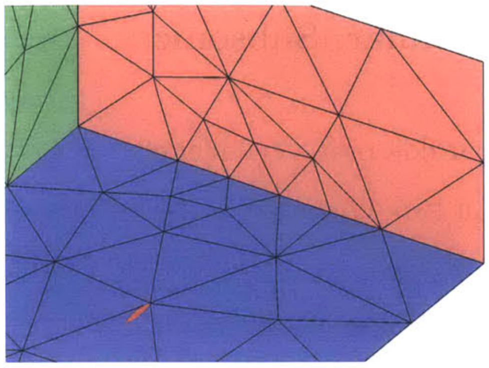
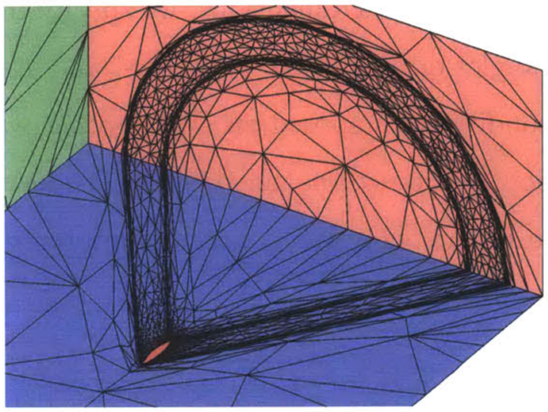


Figure 5-11: Pressure perturbation distribution for "football", inviscid, $M_\infty = 1.8$

Same as the previous cases, the fixed-DOF adaptation is applied, and about 200 adapted meshes are generated and solved. Figure 5-12 shows the initial and the adapted meshes. The initial mesh is very coarse, and the geometry is inside almost one single background element. The adapted meshes for both $p = 1$ and $p = 2$ focus on resolving the shock propagation from the geometry to the outflow boundary. Figure 5-13 shows the adaptation history. For this case where the shock propagation is the only feature for the output prediction, the higher-order discretization is not more efficient at the DOFs considered.


(a) Initial Mesh


(b) Adapted mesh, $p = 1$, $DOF = 160k$

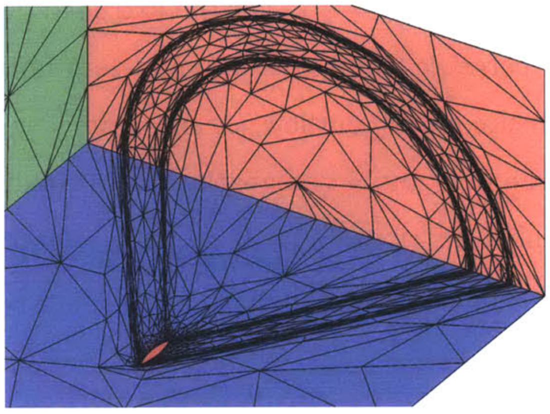
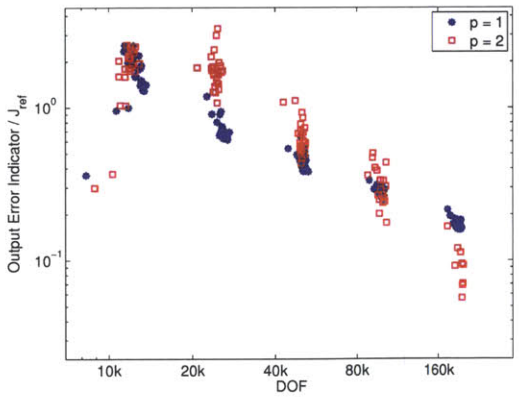
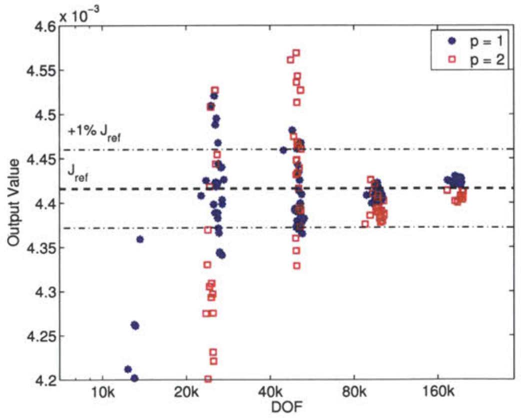

(c) Adapted mesh, $p = 2$, $DOF = 160k$

Figure 5-12: Initial and adapted meshes for "football", inviscid, $M_\infty = 1.8$



(a) Output error indicator



(b) Output value

Figure 5-13: Adaptation history for "football", inviscid, $M_\infty = 1.8$; reference value is obtained at $p = 2$, $DOF = 320k$

5.2.4 Cylinder, Laminar, Subsonic

This case presents a laminar flow over a cylinder of radius r. The setup of this problem is produced by extruding a two-dimensional case into the third dimension by r. The baseline two-dimensional problem has a circle with a radius of r, at the center of a rectangular box, which has a length of $100r$ in the flow direction, and $30r$ in the cross-flow direction. The flow has a Reynolds number of $Re_r = 50$ and a freestream Mach number of $M_\infty = 0.1$. For the extruded problem in three dimensions, we specify total temperature, total pressure, and zero flow angles on the inflow boundary, and static pressure on the outflow. A slip boundary condition is imposed on all the other walls of the box, and an adiabatic no-slip condition is imposed on the cylinder. The Mach number distribution for this case is shown in Figure 5-14, where a recirculation region exists behind the cylinder.

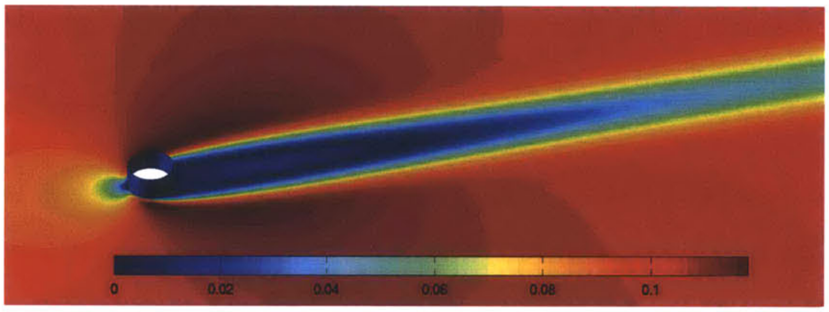
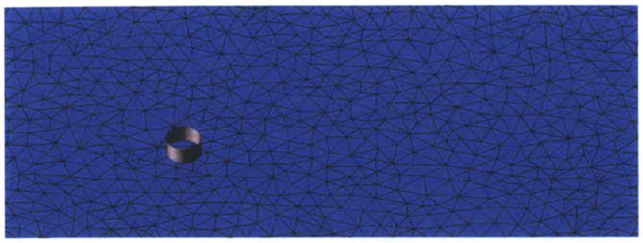
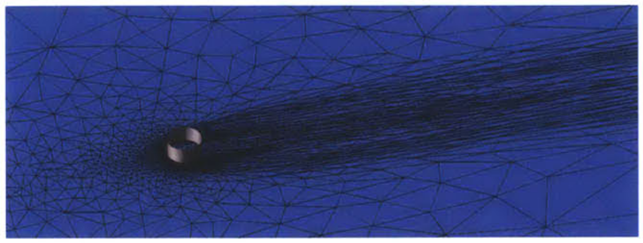


Figure 5-14: Mach number distribution, cylinder, $Re_r = 50$, $M_\infty = 0.1$

The output of interest is drag. Figure 5-15 shows the initial and the adapted meshes. The initial mesh is a uniform mesh, and the adaptation employs anisotropic elements for the boundary layer and the wake. To resolve these smooth features, the $p = 2$ discretization needs many fewer DOFs than $p = 1$. The adaptation history is shown in Figure 5-16, where the $p = 2$ discretization is superior for all DOFs considered. The reference area for computing the drag coefficient is the frontal area.


(a) Initial mesh


(b) Adapted mesh, $p = 1$, $DOF = 80k$

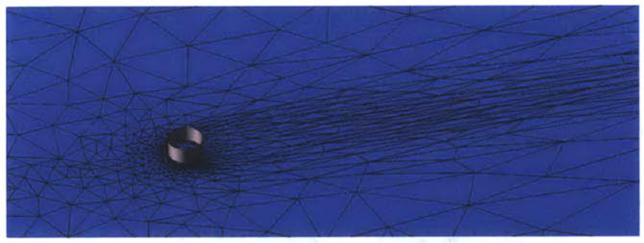
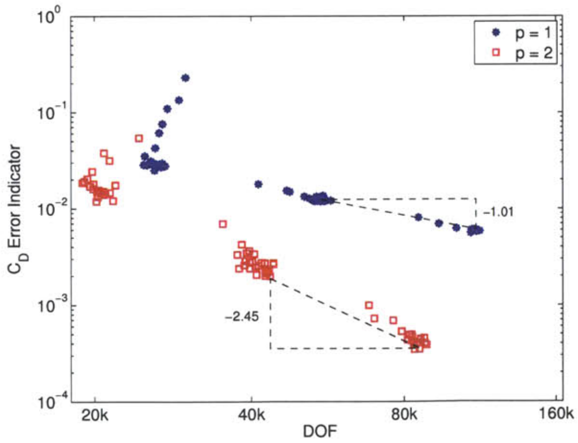
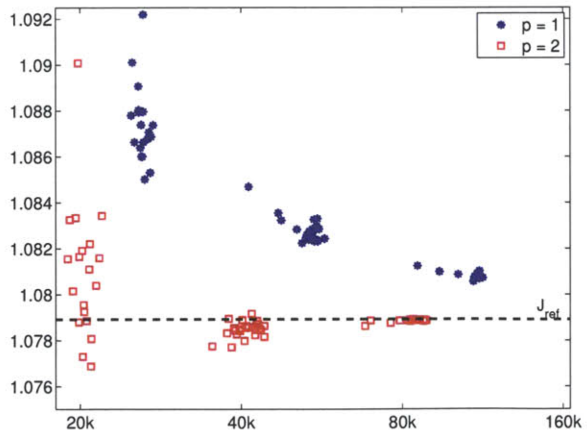

(c) Adapted mesh, $p = 2$, $DOF = 80k$

Figure 5-15: Initial and adapted meshes on the bottom plane, cylinder, $Re_r = 50$, $M_\infty = 0.1$



(a) C_D error indicator



(b) C_D

Figure 5-16: Adaptation history for cylinder, $Re_r = 50$, $M_\infty = 0.1$; reference value obtained from two-dimensional simulations

5.2.5 Body of Revolution (NACA0012), Laminar, Subsonic

This case presents a laminar flow over a geometry produced by revolving the NACA0012 airfoil. Note the airfoil is modified to be closed at $x = 1$. The flow conditions are $Re_c = 5000$, $M_\infty = 0.5$, $\alpha = 1°$, and $\beta = 0$. The output of interest is drag. The Mach number distribution for this case is shown in Figure 5-17. For this case, a total of 150 meshes are generated and solved. Figure 5-18 shows the initial and the adapted meshes. The initial mesh is so coarse that the geometry is inside almost one background element, and the adaptation employs anisotropic elements to resolve the boundary layer and the wake. Figure 5-19 shows the convergence history of the drag coefficient C_D, where the $p = 2$ discretization is superior to $p = 1$. The reference area for C_D calculation is the frontal area.

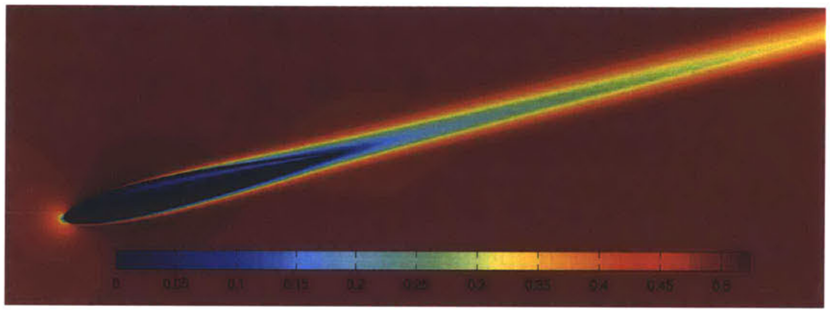
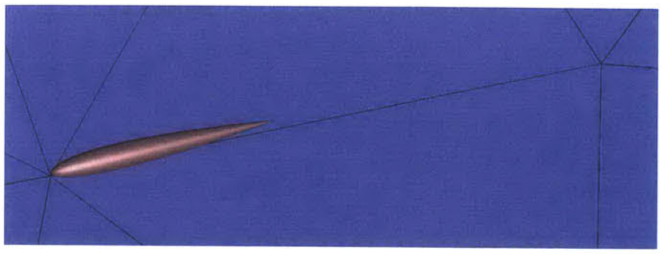
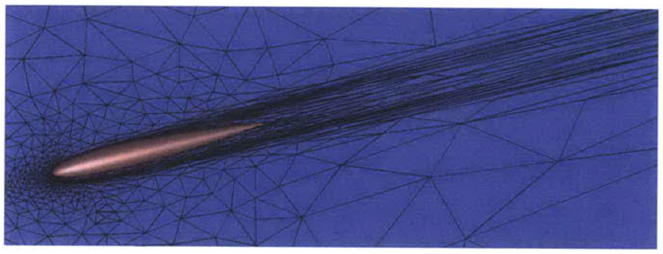


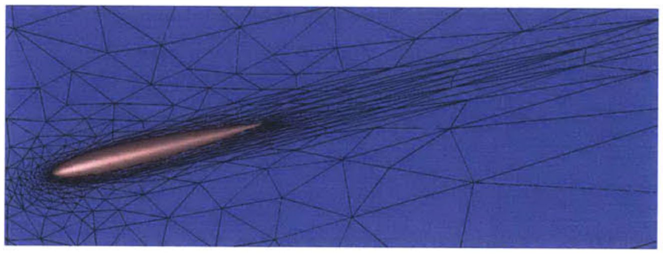
Figure 5-17: Mach number distribution, body of revolution (NACA0012), $Re_c = 5000$, $M_\infty = 0.5$



(a) Initial mesh

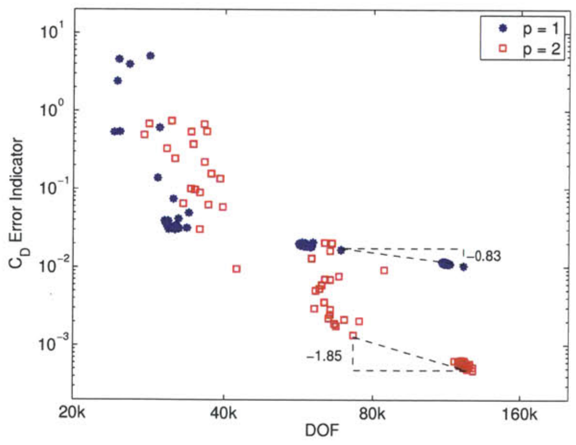


(b) Adapted mesh, $p = 1$, $DOF = 160k$

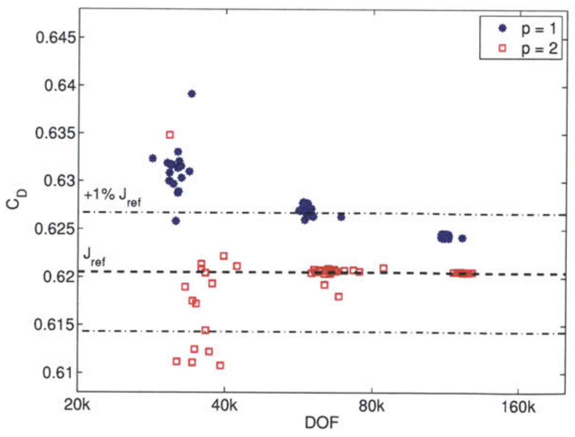


(c) Adapted mesh, $p = 2$, $DOF = 160k$

Figure 5-18: Initial and adapted meshes on the symmetry plane, body of revolution (NACA0012), $Re_c = 5000$, $M_\infty = 0.5$



(a) C_D error indicator



(b) C_D

Figure 5-19: Adaptation history, body of revolution (NACA0012), $Re_c = 5000$, $M_\infty = 0.5$; reference value is obtained at $p = 2$, $DOF = 1.0M$

Chapter 6

Scalar Elliptic Interface Problems

Many engineering applications involve computational domains with multiple materials separated by interfaces of arbitrary shapes. These so-called interface problems are often governed by partial differential equations (PDEs) with discontinuous parameters across the material interfaces, or more generally, by different PDEs across the interfaces. In this chapter, we consider elliptic interface problems defined by

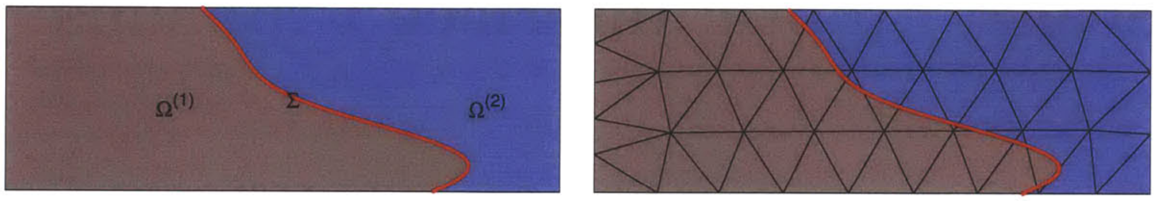
$$-\nabla \cdot (\kappa \nabla u) = f \quad \text{in } \bigcup_{i=1}^{N} \Omega^{(i)}, \tag{6.1}$$
$$u = u_D \quad \text{on } \partial\Omega,$$

where the computational domain $\overline{\Omega} = \bigcup_i \overline{\Omega^{(i)}}$ contains multiple interfaces $\Sigma = \bigcup_{i \neq j} \overline{\Omega^{(i)}} \bigcap \overline{\Omega^{(j)}}$ as illustrated in Figure 6-1(a) for an example of two sub-domains. The coefficient κ can be discontinuous across Σ. In addition, an interface condition (IC) is imposed:

$$[u] = a \quad \text{on } \Sigma, \quad [\kappa \nabla u \cdot \hat{\mathbf{n}}] = b \quad \text{on } \Sigma, \tag{6.2}$$

where $[z] \equiv z^{(-)} - z^{(+)}$ for any scalar field z, and $z^{(-)}$ and $z^{(+)}$ denote the restriction of z on the neighbor sub-domains.

For general interface problems, generating an interface-conforming mesh can be very time-consuming and can require a large amount of human interaction. Many discretization methods have thus been proposed for using non-interface-conforming meshes (also called unfitted meshes), where meshing does not have to conform to the interface and so allows inner-element material discontinuity; an example is illustrated in Figure 6-1(b). However, these methods remain largely at most second-order accurate, and reaching higher-order remains non-trivial.



(a) Example interface problem (b) Example non-interface-conforming mesh

Figure 6-1: Example interface problem and non-conforming mesh

In this chapter, we propose a high-order accurate method for elliptic interface problems on unfitted meshes. Specifically, the solution strategy includes a high-order discontinuous Galerkin (DG) discretization and a simplex cut-cell method. We first derive the DG discretization in a unified form for elliptic interface problems on fitted meshes, and show that no modification on the DG bilinear form is needed for interface treatment. We then extend the method to unfitted meshes using the cut-cell technique, where the interface definition is completely separate from the mesh generation process. No assumption is made on the interface shape (other than Lipschitz continuity). We also combine our strategy with the adaptive scheme presented in Chapter 4 in order to control the effect of possible singularities induced by interface shapes, e.g. corners. Through numerical examples, we demonstrate high-order convergence for elliptic interface problems with both smooth and non-smooth interface shapes. A dual-consistent output evaluation is also derived for the developed DG scheme, and output superconvergence of $\mathcal{O}(h^{2p})$ is observed.

6.1 Overview of Unfitted Methods and High-Order Extensions

In this section, we briefly review the existing unfitted methods for interface problems, and we discuss their potential for high-order extensions. These methods can be classified based on how the interface condition (IC) is imposed. Specifically, we classify the methods as follows: 1) finite difference with different treatments of IC, 2) finite element with IC *strongly* imposed, and 3) finite element and finite volume with IC *weakly* imposed. We then introduce the cut-cell method as one belonging to the third type in our classification, and highlight its potential for convenient extension to high-order accuracy for complex interface problems.

Finite Difference

Finite difference schemes have been proposed when the interface Σ lies between grid nodes. Peskin proposed the immersed boundary method (IBM) [107], where the boundary of an immersed object is treated as a singular force along the boundary; see [99] for a review of its applications. Second-order accuracy is shown in one dimension [20]. LeVeque and Li [73] then developed a second-order accurate method for higher dimensions, namely the immersed interface method (IIM), in which the interface jump condition (6.2) is incorporated into the local Taylor expansion; see [76] for a review. Another popular approach is the ghost fluid method (GFM) [47], which is also generally first-order. The key idea of the method is to extrapolate solutions on one side of the interface into ghost cells or fictitious nodes on the other side.

Since the initial publications of the IIM and GFM methods, there have been several researchers developing high-order extensions to these methods. One natural high-order extension to the IIM method is to use (or approximate) the jump interface conditions in higher derivatives of u [77, 133]. High-order methods using fictitious nodes were also developed for elliptic interface problems [52, 134]. While these methods are at least fourth-order accurate for smooth problems, they require a large stencil, and there has been little progress in rigorous stability or convergence proofs for these methods on interface problems.

Finite Element, Strong Imposition of IC

The immersed interface method has been extended to finite element discretizations, and is often named the immersed finite element (IFE) method. The interface condition (6.2) is strongly imposed by modifying the basis functions. In one through three dimensions, (6.2) is sufficient in defining a unique linear nodal basis as demonstrated in [67, 74, 75], and the optimal *a priori* convergence rate is proven in each dimension.

Extending the IFE method to high order is intrinsically difficult. The first difficulty lies in the fact that condition (6.2) alone is not sufficient in defining a unique high-order basis [27]. Different constraints on the high-order basis functions have been proposed [1, 27], mainly for one dimension. While it was observed that some choices of the constraints lead to suboptimal convergence, it is unclear how to systematically choose a correct set of constraints. The second difficulty is in constructing a basis function when Σ has a complex shape in one element. The previous works for two and three dimensions (e.g. [67, 74]) need to assume certain shapes on Σ, for example intersecting each triangle only twice. When these assumptions are not met, it is difficult, if not impossible, to construct even a linear basis.

Finite Element/Finite Volume, Weak Imposition of IC

Another approach for interface problems is to weakly impose the interface condition (6.2) while allowing the elements not to conform to Σ. Many methods belong to this group, for example, a penalty method to impose (6.2) [8, 12, 58], a Lagrange multiplier method for an embedded boundary condition [54], and a mortar finite element method [62]. Note for the last example, interface-conforming meshes are still required, but they do not have to match from the two sides of the interface. These ideas are the same as enforcing a Dirichlet boundary condition through penalties for elliptic boundary value problems; see [7] for example. A second-order finite volume method using Cartesian cut cells has also been developed for elliptic interface problems [64], where the interface condition is imposed through numerical fluxes.

With the interface condition imposed weakly, only an integration along the interface is needed. As a result, elements of arbitrary shapes are allowed in the scheme, provided that integration can be performed on these shapes. Furthermore, no constraint from the interface condition needs to be imposed on the finite element basis functions, and this facilitates the extension to a high-order basis in any dimension. However, there has been little development on high-order finite element method for interface problems with arbitrary interface shapes in two and three dimensions.

As the cut-cell technique has shown success for problems with complex boundaries (see for example [2, 50, 69, 87, 131]), we extend the simplex cut-cell technique to solving interface problems in our work. In particular, we combine the technique with a high-order DG discretization, which easily allows weak imposition of the interface condition (6.2). The key features of our solution strategy include:

- arbitrarily shaped elements enabled to handle complex interfaces;
- easy extension to high order;
- arbitrary anisotropy permitted by simplex elements.

6.2 Discontinuous Galerkin Method for Interface Problems

In this section, we first derive the DG discretization in a unified form for elliptic interface problems on fitted meshes. The consistency of the unified form is then proven by imposing certain constraints on the numerical fluxes. We then choose our numerical fluxes that satisfy these constraints. More specifically, we use the second form of Bassi and Rebay (BR2) [17] for non-interface faces, and the construction of numerical fluxes for interface faces is inspired by the work of Guyomarc'h and Lee [56].

For these choices, we show that in the bilinear form, the interface faces are in fact treated exactly the same as for non-interface faces. At the end of this section, we prove the stability and the optimal convergence of the scheme.

6.2.1 Notations

To derive a unified DG formulation for elliptic interface problems defined in Eq. (6.1) and (6.2), we follow the notations and derivations in the work of Arnold *et al.* [7]. For simplicity of presentation, we assume a homogeneous Dirichlet boundary condition is imposed on $\partial\Omega$ in this section, i.e. $u_D = 0$. Define a triangulation $\mathcal{T}_h$ of the domain Ω into non-overlapping elements K of characteristic size h. Denote all element faces by $\Gamma_A \equiv \bigcup_K \partial K$, and let the mesh be fitted, i.e. $\Sigma \subset \Gamma_A$. Denote the set of non-interface faces by $\Gamma \equiv \Gamma_A \backslash \Sigma$, and the set of non-interface interior faces by $\Gamma_I \equiv \Gamma_A \backslash (\Sigma \cup \partial\Omega)$. Further, we denote the set of faces of all elements in $\Omega^{(i)}$ by $\Gamma_A^{(i)} \equiv \bigcup_{K \subset \Omega^{(i)}} \partial K$, and the interfaces on the boundary of $\Omega^{(i)}$ by $\Sigma^{(i)} \equiv \Sigma \bigcap \partial\Omega^{(i)}$. We then define $\Gamma^{(i)} \equiv \Gamma_A^{(i)} \backslash \Sigma^{(i)}$ and $\Gamma_I^{(i)} \equiv \Gamma_A^{(i)} \backslash \partial\Omega^{(i)}$.

We define the space

$$V \equiv \bigoplus_i H^1(\Omega^{(i)}),$$

and for DG discretization, we denote a finite-dimensional approximation space $V_{h,p}$ on $\mathcal{T}_h$ by

$$V_{h,p} \equiv \{v \in L^2(\Omega) : v|_K \in \mathcal{P}^p(K),\ \forall K \in \mathcal{T}_h\},$$

and similarly, denote

$$V_{h,p}^{(i)} \equiv \{v \in L^2(\Omega^{(i)}) : v|_K \in \mathcal{P}^p(K),\ \forall K \subset \Omega^{(i)}\}.$$

In addition, the jump operator $[\![\cdot]\!]$ and average operator $\{\cdot\}$ follow the definitions in [7]. Specifically, on a non-boundary face $e \in \Gamma_A \backslash \partial\Omega$, we define for an arbitrary scalar function x and vector function $\mathbf{y}$:

$$\begin{aligned} [\![x]\!] &\equiv x^{-}\mathbf{n}^{-} + x^{+}\mathbf{n}^{+}, & [\![\mathbf{y}]\!] &\equiv \mathbf{y}^{-}\cdot\mathbf{n}^{-} + \mathbf{y}^{+}\cdot\mathbf{n}^{+}, \\ \{x\} &\equiv \frac{1}{2}(x^{-} + x^{+}), & \{\mathbf{y}\} &\equiv \frac{1}{2}(\mathbf{y}^{-} + \mathbf{y}^{+}), \end{aligned}$$

where $(\cdot)^{-}$ and $(\cdot)^{+}$ denote the trace values on e of any quantity $(\cdot)$ evaluated for the neighbor elements, K^{-} and K^{+}, and $\hat{\mathbf{n}}^{-}$ and $\hat{\mathbf{n}}^{+}$ are the unit normal vector pointing exterior to K^{-} and K^{+}, respectively. On a boundary face $e \in \partial\Omega$, we define

$$\begin{aligned} [\![x]\!] &\equiv x\hat{\mathbf{n}}, & [\![\mathbf{y}]\!] &\equiv \mathbf{y}\cdot\hat{\mathbf{n}} \\ \{x\} &= x, & \{\mathbf{y}\} &= \mathbf{y}, \end{aligned}$$

where $\hat{\mathbf{n}}$ denotes the unit normal vector pointing exterior to Ω. Also, for any function $q \in V + V_{h,p}$, $q^{(i)}$ denotes its restriction on $\Omega^{(i)}$.

6.2.2 Mixed and Primal Formulations

We first rewrite Eq. (6.1) on each $\Omega^{(i)}$ as two first-order equations:

$$\sigma^{(i)} = \nabla u^{(i)} \tag{6.3}$$

$$-\nabla\cdot\left(\kappa^{(i)}\sigma^{(i)}\right) = f^{(i)}. \tag{6.4}$$

Let $u_h \in V_{h,p}$ and $\sigma_h \in [V_{h,p}]^d$ denote the DG solutions, and we follow the same derivation in [7] but for each $\Omega^{(i)}$ separately, then Eq. (6.3) and (6.4) lead to the mixed formulation on each $\Omega^{(i)}$:

$$\int_{\Omega^{(i)}} \sigma_h^{(i)} \cdot \tau d\Omega = -\int_{\Omega^{(i)}} u_h^{(i)} \nabla \cdot \tau d\Omega + \int_{\Gamma^{(i)}} [\![\hat{u}^{(i)}]\!] \cdot \{\tau\} ds + \int_{\Gamma_I^{(i)}} \{\hat{u}^{(i)}\} [\![\tau]\!] ds + \int_{\Sigma^{(i)}} \hat{u}^{(i)} \tau \cdot \hat{\mathbf{n}}^{(i)} ds \tag{6.5}$$

$$\int_{\Omega^{(i)}} \kappa^{(i)} \sigma_h^{(i)} \cdot \nabla v d\Omega - \int_{\Gamma^{(i)}} \{\widehat{\kappa\sigma}^{(i)}\} \cdot [\![v]\!] ds - \int_{\Gamma_I^{(i)}} [\![\widehat{\kappa\sigma}^{(i)}]\!] \{v\} ds - \int_{\Sigma^{(i)}} v \widehat{\kappa\sigma}^{(i)} \cdot \hat{\mathbf{n}}^{(i)} ds = \int_{\Omega^{(i)}} f^{(i)} v d\Omega, \tag{6.6}$$

for arbitrary test functions $\tau \in [V_{h,p}^{(i)}]^d$ and $v \in V_{h,p}^{(i)}$, where $\hat{\mathbf{n}}^{(i)}$ is the unit normal vector on $\Sigma^{(i)}$ pointing out of $\Omega^{(i)}$, and $\hat{u}$ and $\widehat{\kappa\sigma}$ represent the numerical fluxes. On the interface $\Sigma^{(i)}$, $\hat{u}^{(i)}$ and $\widehat{\kappa\sigma}^{(i)}$ denote the numerical fluxes evaluated from the side of $\Omega^{(i)}$.

To derive the primal formulation, we first apply integration by parts on the right-hand side of (6.5):

$$\int_{\Omega^{(i)}} \sigma_h^{(i)} \cdot \tau d\Omega = \int_{\Omega^{(i)}} \nabla u_h^{(i)} \cdot \tau d\Omega + \int_{\Gamma^{(i)}} [\![\hat{u}^{(i)} - u_h^{(i)}]\!] \cdot \{\tau\} ds + \int_{\Gamma_I^{(i)}} \{\hat{u}^{(i)} - u_h^{(i)}\} [\![\tau]\!] ds + \int_{\Sigma^{(i)}} (\hat{u}^{(i)} - u_h^{(i)}) \tau \cdot \hat{\mathbf{n}}^{(i)} ds, \quad \forall \tau \in \left[V_{h,p}^{(i)}\right]^d. \tag{6.7}$$

We then let $\tau = \kappa^{(i)} \nabla v$ in (6.7), and substituting into (6.6) gives the primal formulation on each $\Omega^{(i)}$:

$$\begin{aligned} \int_{\Omega^{(i)}} \nabla u_h^{(i)} \cdot \kappa^{(i)} \nabla v d\Omega &+ \int_{\Gamma^{(i)}} \left([\![\hat{u}^{(i)} - u_h^{(i)}]\!] \cdot \{\kappa^{(i)} \nabla v\} - \{\widehat{\kappa\sigma}^{(i)}\} \cdot [\![v]\!] \right) ds \\ &+ \int_{\Gamma_I^{(i)}} \left(\{\hat{u}^{(i)} - u_h^{(i)}\} [\![\kappa^{(i)} \nabla v]\!] - [\![\widehat{\kappa\sigma}^{(i)}]\!] \{v\} \right) ds \\ &+ \int_{\Sigma^{(i)}} \left((\hat{u}^{(i)} - u_h^{(i)}) \kappa^{(i)} \nabla v - \widehat{\kappa\sigma}^{(i)} v \right) \cdot \hat{\mathbf{n}}^{(i)} ds = \int_{\Omega^{(i)}} f^{(i)} v d\Omega, \quad \forall v \in V_{h,p}^{(i)}. \end{aligned} \tag{6.8}$$

We then sum Eq. (6.8) over i, and the interface term $\int_\Sigma (\cdot) ds$ can be combined into the non-interface terms. This gives the discretized weak form of the interface problem on Ω: find $u_h \in V_{h,p}$ such that

$$B_{h,p}(u_h, v) = l_{h,p}(v), \quad \forall v \in V_{h,p} \tag{6.9}$$

where

$$\begin{aligned} B_{h,p}(u_h, v) = &\int_\Omega \nabla u_h \cdot \kappa \nabla v d\Omega + \int_{\Gamma_A} \left([\![\hat{u} - u_h]\!] \cdot \{\kappa \nabla v\} - \{\widehat{\kappa\sigma}\} \cdot [\![v]\!] \right) ds \\ &+ \int_{\Gamma_A \backslash \partial\Omega} \left(\{\hat{u} - u_h\} [\![\kappa \nabla v]\!] - [\![\widehat{\kappa\sigma}]\!] \{v\} \right) ds \qquad (6.10) \\ l_{h,p}(v) = &\int_\Omega f v d\Omega. \qquad (6.11) \end{aligned}$$

To complete the primal formulation, we still need to express σ_h in terms of u_h We sum (6.7) over i, and define the lifting operators $r : [L^2(\Gamma_A)]^d \to [V_{h,p}]^d$ and $l : L^2(\Gamma_A \backslash \partial\Omega) \to [V_{h,p}]^d$:

$$\int_\Omega r(\phi) \cdot \tau d\Omega = -\int_{\Gamma_A} \phi \cdot \{\tau\} ds, \qquad \int_\Omega l(q) \cdot \tau d\Omega = -\int_{\Gamma_A \backslash \partial\Omega} q [\![\tau]\!] ds, \quad \forall \tau \in [V_{h,p}]^d, \tag{6.12}$$

then we obtain

$$\sigma_h = \nabla u_h - r([\![\hat{u} - u_h]\!]) - l(\{\hat{u} - u_h\}). \tag{6.13}$$

6.2.3 Primal Consistency

Let u be the solution to Eq. (6.1) and satisfy the interface condition Eq. (6.2). Let the numerical flux $\hat{u}$ satisfy

$$[\![\hat{u} - u]\!] = 0, \quad \{\hat{u} - u\} = 0 \quad \text{on } e \in \Gamma_A, \tag{6.14}$$

then $\sigma_h(u) = \nabla u$ from Eq. (6.13). If we further let $\widehat{\kappa\sigma}$ satisfy

$$[\![\widehat{\kappa\sigma} - \kappa\sigma]\!] = 0, \quad \{\widehat{\kappa\sigma} - \kappa\sigma\} = 0 \quad \text{on } e \in \Gamma_A, \tag{6.15}$$

then substituting the conditions (6.14) and (6.15) into (6.10) can easily prove the consistency of the scheme. Also, note the jump conditions in (6.14) and (6.15) on Σ are equivalent to

$$[\![\hat{u}]\!] = a\hat{\mathbf{n}}^{(-)}, \quad [\![\widehat{\kappa\sigma}]\!] = b \quad \text{on } \Sigma, \tag{6.16}$$

which weakly imposes the interface condition Eq. (6.2).

6.2.4 Numerical Fluxes and Final Discretized Form

Eq. (6.9) defines a family of DG schemes, where consistency is guaranteed when $\hat{u}$ and $\widehat{\kappa\sigma}$ satisfy (6.14) and (6.15) as discussed. In this section, we present one choice of consistent numerical fluxes. On non-interface faces, we use BR2 [17] for numerical fluxes:

$$\hat{u} = \{u_h\}, \qquad \widehat{\kappa\sigma} = \{\kappa[\nabla u_h + \eta_e r^e([\![u_h]\!])]\}, \qquad \text{on } e \in \Gamma_I, \tag{6.17}$$

$$\hat{u} = u_D = 0, \qquad \widehat{\kappa\sigma} = \kappa[\nabla u_h + \eta_e r^e([\![u_h]\!])], \qquad \text{on } e \in \partial\Omega, \tag{6.18}$$

where η_e is a positive number on each face $e \in \Gamma_A$, and $r^e(\cdot)$ is the local lifting operator defined by

$$\int_{\Omega_h^e} r^e(\phi) \cdot \tau d\Omega = -\int_e \phi \cdot \{\tau\} ds, \quad \forall \tau \in [V_{h,p}]^d, \tag{6.19}$$

and Ω_h^e is the union of elements sharing the face e. On interface faces, $e \in \Sigma$, we need to modify Eq. (6.17) in order to satisfy condition (6.16). One natural choice is

$$\begin{cases} \hat{u}^{(-)} & = \{u_h\} + \frac{1}{2}a \\ \hat{u}^{(+)} & = \{u_h\} - \frac{1}{2}a \end{cases} \qquad \begin{cases} \widehat{\kappa\sigma}^{(-)} & = \{\kappa[\nabla u_h + \eta_e r^e([\![u_h]\!] - a\hat{\mathbf{n}}^{(-)})]\} + \frac{1}{2}b\hat{\mathbf{n}}^{(-)} \\ \widehat{\kappa\sigma}^{(+)} & = \{\kappa[\nabla u_h + \eta_e r^e([\![u_h]\!] - a\hat{\mathbf{n}}^{(-)})]\} - \frac{1}{2}b\hat{\mathbf{n}}^{(-)} \end{cases},$$

where the terms $\pm\frac{1}{2}a$ and $\pm\frac{1}{2}b\hat{\mathbf{n}}^{(1)}$ are also proposed in [56] for a particular form of local DG method on interface problems, and $r^e(\cdot)$ is defined in (6.19). Substituting these numerical fluxes into Eq. (6.9) gives the final discretized form:

$$\begin{aligned} B_{h,p}(u_h, v) = &\int_\Omega \nabla u_h \cdot \kappa \nabla v d\Omega - \int_{\Gamma_A} ([\![u_h]\!] \cdot \{\kappa\nabla v\} + \{\kappa\nabla u_h\} \cdot [\![v]\!])\, ds \\ &- \sum_{e\in\Gamma_A} \int_e \eta_e \{\kappa r^e([\![u_h]\!])\} \cdot [\![v]\!] ds \end{aligned} \tag{6.20}$$

$$l_{h,p}(v) = \int_\Omega f v d\Omega - \int_\Sigma \left(a\{\kappa\nabla v\} \cdot \hat{\mathbf{n}}^{(1)} - b\{v\}\right) ds - \sum_{e\in\Sigma} \int_e \eta_e \{\kappa r^e(a\hat{\mathbf{n}}^{(-)})\} \cdot [\![v]\!] ds, \tag{6.21}$$

where the bilinear form treats the interface faces exactly the same as non-interface faces.

6.2.5 Optimal Convergence

Our choice of numerical fluxes leads to consistency of the scheme as discussed in the previous sections. Further, because our bilinear form in Eq. (6.20) is identical to a case without interface, the proof for boundedness and stability is very similar to the proof in [7], and is shown in Appendix G.1. Optimal convergence then follows from the consistency and stability, see [7].

6.3 Dual Consistency and Output Superconvergence

In this section, we prove a dual-consistent evaluation for common outputs, and the dual consistency leads to output superconvergence for the developed DG scheme on interface problems. Let $u_{h,p} \in V_{h,p}$ denote the DG solution, which satisfies

$$B_{h,p}(u_{h,p}, v) - l_{h,p}(v) = 0, \quad \forall v \in V_{h,p},$$

where $B_{h,p}(\cdot,\cdot)$ and $l_{h,p}(\cdot)$ are defined in Eq. (6.20) and (6.21), respectively. We are interested in the output defined by

$$J = \mathcal{J}(u) \equiv \sum_i \int_{\Omega^{(i)}} g_\Omega^{(i)} u^{(i)} d\Omega - \int_{\partial\Omega} g_{\partial\Omega} \kappa \nabla u \cdot \hat{\mathbf{n}} ds - \int_\Sigma g_\Sigma \kappa^{(-)} \nabla u^{(-)} \cdot \hat{\mathbf{n}}^{(-)} ds, \tag{6.22}$$

where g_Ω, $g_{\partial\Omega}$, and g_Σ are functions defined on Ω, $\partial\Omega$, and Σ, respectively. Denote the approximate output value computed from $u_{h,p}$ by $J_{h,p} = \mathcal{J}_{h,p}(u_{h,p}) \equiv \mathcal{J}^l_{h,p}(u_{h,p}) + \mathcal{J}^o_{h,p}$, where $\mathcal{J}^l_{h,p}(\cdot) : V_{h,p} \to \mathbb{R}$ is a linear functional. For a more accurate prediction of the output value, the discrete evaluation $\mathcal{J}_{h,p}(u_{h,p})$ needs to be dual consistent, i.e. the exact dual solution, $\psi \in W \equiv V + V_{h,p}$, needs to satisfy the discrete adjoint equation:

$$B_{h,p}(w, \psi) - \mathcal{J}^l_{h,p}(w) = 0, \quad \forall w \in W. \tag{6.23}$$

When Eq. (6.23) is satisfied, we can observe superconvergence for the output value as proven in [53, 82].

In this work, we evaluate the output in Eq. (6.22) by

$$\begin{aligned}\mathcal{J}_{h,p}(u_{h,p}) = &\sum_i \int_{\Omega^{(i)}} g_\Omega^{(i)} u_{h,p}^{(i)} d\Omega - \sum_{e\in\partial\Omega} \int_e g_{\partial\Omega} \kappa [\nabla u_{h,p} + \eta_e r^e([\![u_{h,p}]\!])] \cdot \hat{\mathbf{n}} ds \\ &- \sum_{e\in\Sigma} \int_e g_\Sigma \{\kappa[\nabla u_{h,p} + \eta_e r^e([\![u_{h,p}]\!] - a\hat{\mathbf{n}}^{(-)})]\} \cdot \hat{\mathbf{n}}^{(-)} ds - \frac{1}{2}\int_\Sigma g_\Sigma b ds,\end{aligned} \tag{6.24}$$

where the last term is for primal consistency, and η_e and $r^e(\cdot)$ are defined in Section 6.2.4. To prove dual consistency of $\mathcal{J}_{h,p}(u_{h,p})$ defined in (6.24), we first derive the continuous dual problem, which is found to be

$$-\nabla \cdot (\kappa \nabla \psi) = g_\Omega^{(i)} \text{ in } \Omega^{(i)}, \quad \forall i \tag{6.25}$$

$$\psi = g_{\partial\Omega} \text{ on } \partial\Omega \tag{6.26}$$

$$[\![\psi]\!] = g_\Sigma \hat{\mathbf{n}}^{(-)}, \quad [\![\kappa \nabla \psi]\!] = 0 \text{ on } \Sigma. \tag{6.27}$$

Detailed derivation is in Appendix G.2. Then from Eq. (6.20), we have

$$\begin{aligned} B_{h,p}(w,\psi) = &\int_\Omega \nabla w \cdot \kappa \nabla \psi d\Omega - \int_{\Gamma_A} [\![w]\!] \cdot \{\kappa \nabla \psi\} + \{\kappa \nabla w\} \cdot [\![\psi]\!] ds \\ &- \sum_{e \in \Gamma_A} \int_e \eta_e \{\kappa r^e([\![w]\!])\} \cdot [\![\psi]\!] ds, \end{aligned} \tag{6.28}$$

where ψ is solution to Eq. (6.25) through (6.27), and so satisfies

$$\begin{gathered} \psi = g_{\partial\Omega} \text{ on } e \in \partial\Omega, \\ [\![\psi]\!] = 0 \text{ on } e \in \Gamma_I, \quad [\![\psi]\!] = g_\Sigma \hat{\mathbf{n}}^{(-)} \text{ on } e \in \Sigma. \end{gathered}$$

Thus, applying integration by parts to the first term in (6.28) leads to

$$\begin{aligned} B_{h,p}(w,\psi) = &- \int_\Omega w \nabla \cdot (\kappa \nabla \psi) - \int_{\partial\Omega} \kappa \nabla w \cdot g_{\partial\Omega} \hat{\mathbf{n}} ds - \sum_{e \in \partial\Omega} \int_e \eta_e \kappa r^e([\![w]\!]) \cdot g_{\partial\Omega} \hat{\mathbf{n}} ds \\ &- \int_\Sigma \{\kappa \nabla w\} \cdot g_\Sigma \hat{\mathbf{n}}^{(-)} ds - \sum_{e \in \Sigma} \int_e \eta_e \{\kappa r^e([\![w]\!])\} \cdot g_\Sigma \hat{\mathbf{n}}^{(-)} ds. \end{aligned} \tag{6.29}$$

Also, from Eq. (6.24), we have

$$\begin{aligned} \mathcal{J}_{h,p}^l(w) = &\sum_i \int_{\Omega^{(i)}} g_\Omega^{(i)} w^{(i)} d\Omega - \sum_{e \in \partial\Omega} \int_e \kappa [\nabla w + \eta_e r^e([\![w]\!])] \cdot g_{\partial\Omega} \hat{\mathbf{n}} ds \\ &- \sum_{e \in \Sigma} \int_e \{\kappa [\nabla w + \eta_e r^e([\![w]\!])]\} \cdot g_\Sigma \hat{\mathbf{n}}^{(-)} ds. \end{aligned} \tag{6.30}$$

Combining Eq. (6.29) and (6.30) gives

$$B_{h,p}(w,\psi) - \mathcal{J}^{l}_{h,p}(w) = -\sum_i \int_{\Omega^{(i)}} w^{(i)} \left[g_\Omega^{(i)} + \nabla \cdot \left(\kappa^{(i)} \nabla \psi^{(i)} \right) \right] d\Omega = 0,$$

where the last equality is because of (6.25), and this proves the dual consistency.

6.4 Cut-Cell Technique for Interface Problems

On a cut-cell mesh, the mesh generation process does not conform to the interface, i.e. the interface definition is completely separate from the *background* mesh, which is denoted by $\mathcal{T}_{h,b}$ consisting of elements K_b. The interface then intersects with $\mathcal{T}_{h,b}$, and cuts $\mathcal{T}_{h,b}$ into separate parts $\{\mathcal{T}_h^{(i)}\}$, each of which completely lies inside one $\Omega^{(i)}$. For example, Figure 6-2 shows a background element K_b cut by the interface into two elements, $K^{(1)}$ and $K^{(2)}$, lying completely in $\Omega^{(1)}$ and $\Omega^{(2)}$, respectively. The mesh after the cutting process is named a cut mesh, and denoted by $\mathcal{T}_h = \cup_i \mathcal{T}_h^{(i)}$, which consists of elements of arbitrary shapes along the interface.

The DG discretization derived in Section 6.2 is applied on the cut mesh $\mathcal{T}_h$, and the choice of the DG space for cut elements was presented in Section 4.2. While the derived DG method has no assumption on physical dimensions, in this work we consider two-dimensional interface problems. The mechanism of constructing a two-dimensional cut mesh, as well as the quadrature rule generation, will be briefly described in the rest of this section. Throughout this work, no assumption is made on the interface shape (other than Lipschitz continuity). Note the techniques developed in Chapters 2 and 3 for three dimensions are also applicable to interface problems, but were not implemented in two dimensions at the time of writing this chapter.

In this work, the interface geometry for two-dimensional problems is represented by piecewise cubic splines as in [50, 87], and the orientation of each spline defines normal vectors pointing into one $\Omega^{(i)}$. Given a cubic spline representation and a simplex background mesh, the intersection algorithm constructs the topology of the cut mesh

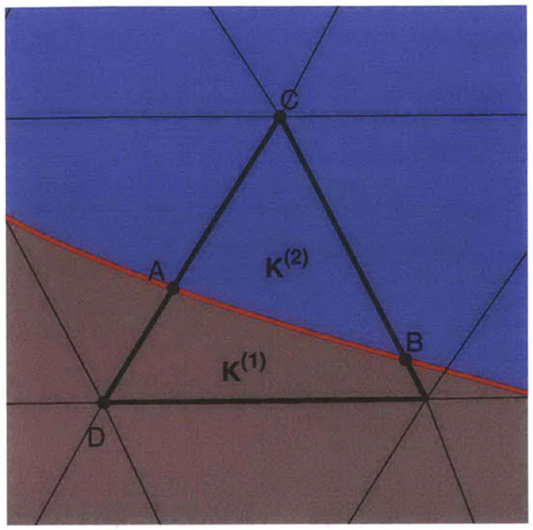


Figure 6-2: Example of background elements cut into two elements, $K^{(1)}$ and $K^{(2)}$

$\mathcal{T}_h$. The algorithm consists of first solving for all the intersection points between the splines and every background edge, for example, points A and B in Figure 6-2. Then the cut background edges are constructed, e.g. AC and AD in Figure 6-2. For each cut background edge, we determine in which $\Omega^{(i)}$ it lies, based on the orientation of the splines. The final step consists of connecting the cut background edges and cut splines into cut elements. The details of the intersection algorithm can be found in the work of Modisette [87], where the main difference from this work is that Modisette constructs cut elements on only one side of the interface, which is in fact the domain boundary in his work.

As a cut-cell mesh can have arbitrarily shaped elements, a quadrature rule for each of these elements and their faces is required in the DG discretization defined in Eq. (6.20) and (6.21). For a cut element K in two dimensions, the quadrature points $\mathbf{x}_q$ are chosen to be equidistant points in the oriented bounding box of K but lying inside K. An example is shown in Figure 6-3. The quadrature weights $\mathbf{w}_q$ are calculated using the algorithm proposed in [50], which is based on minimization of projection error on $\mathbf{x}_q$'s. In addition, because most cut elements have three or four sides, they can often be mapped to triangles or quadrilaterals, respectively, using high-order Lagrange basis, as proposed in [87]. We can then employ standard integration rules for the cut elements that are converted to these "canonical" shapes; and this integration procedure is shown to be more efficient [87]. An example cut element that can be converted to a high-order triangle is shown in Figure 6-4 together with the quadrature points. Note when an element K on the interface Γ is converted to "canonical" by a mapping g_K, the interface geometry on ∂K is in fact redefined according to g_K, and may not be the same as the original geometry. The information of this redefinition has to be shared with the neighbors of K, so that two neighbor elements still define the same common face.

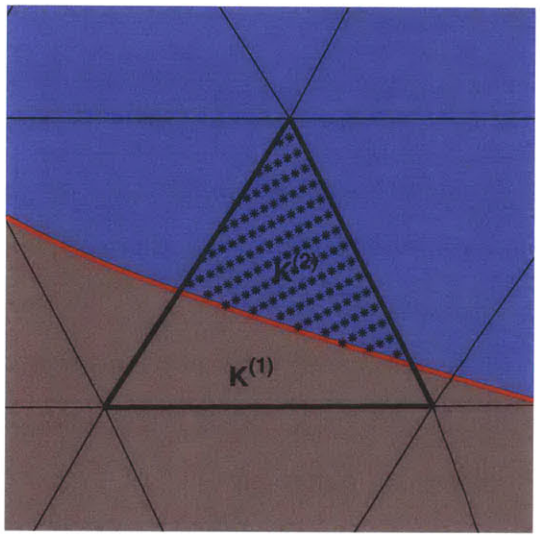


Figure 6-3: Example cut element with con- equidistant quadrature points in the oriented bounding box

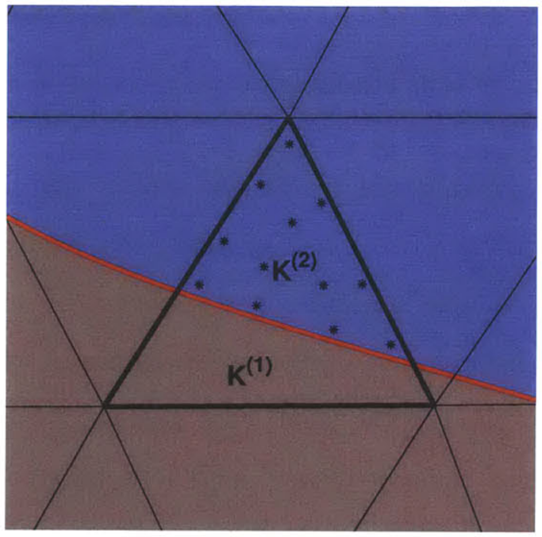


Figure 6-4: Example cut element converted to canonical element, with canonical quadrature points

6.5 Results

In this section, we present four numerical examples that confirm the optimal order of convergence for our scheme. The first two examples have a smooth interface, where we demonstrate the optimal L^2-error convergence, and the superconvergence of diffusive flux error. The latter two examples have solution singularities induced by the interface shapes. We demonstrate that the effect of singularities can be controlled by mesh adaptation, and output superconvergence can be achieved despite the present singularities. Moreover, the last example presents a case of reaction-diffusion equation, where an anisotropic feature exists along the interface. Note the adaptation scheme is the MOESS algorithm presented in Section 4.4.

Example 1: L^2-Error Convergence

This example is from [64], where the interface geometry is defined in polar coordinates by $r(\theta) = 0.32+0.05\cos(6\theta)$, and the computational domain is $[-0.75, 0.75] \times [-0.75, 0.75]$ as shown in Figure 6-5(a). The diffusivities are given by $\kappa^{(1)} = 1$ and $\kappa^{(2)} = 10$. The source function, the boundary conditions, and the interface condition defined in Eq. (6.2) are imposed such that we have an exact solution given by

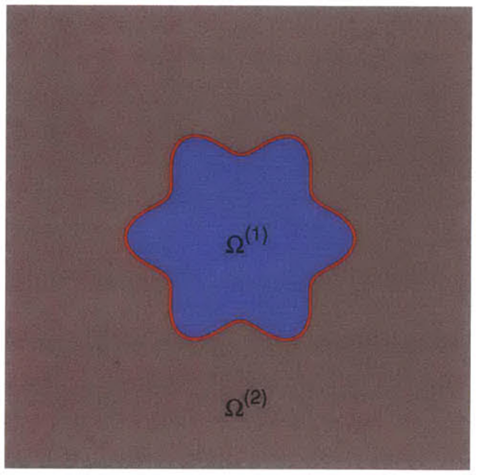
$$u(x,y) = \begin{cases} e^x(x^2\sin y + y^2) & \text{in } \Omega^{(1)} \\ -(x^2+y^2) & \text{in } \Omega^{(2)} \end{cases}, \tag{6.31}$$

which is shown in Figure 6-5(b). This example is then solved on a sequence of uniformly refined background meshes, the coarsest of which is shown in Figure 6-6. Figure 6-7 shows the L^2-error convergence for $p = 1$ through 4, and we do observe the optimal convergence rate h^{p+1} for each p. Note that the error reaches machine precision level on the finest mesh using $p = 4$.

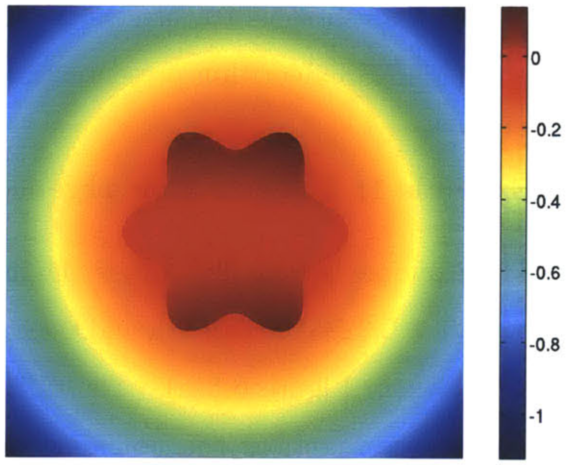
Example 2: Diffusive Flux Error Convergence

This example demonstrates error convergence for heat flux defined in Eq. (6.22) with $g_\Omega = 0$ and $g_{\partial\Omega} = 0$, and confirms the discrete evaluation in Eq. (6.24) is dual-consistent. The interface geometry and computational domain are the same as in Example 1, and the exact solution u is also given by Eq. (6.31).

We let $g_\Sigma = \sin 2\theta$, where θ represents the angle with respect to the positive x-axis. Figure 6-8 shows the adjoint solution for this case. The output-based adaptive scheme is then applied for $p = 1$ and $p = 3$ at four different DOFs: $2k$, $4k$, $8k$, and



(a) Computational domain



(b) Primal solution

Figure 6-5: Computational domain and primal solution of Example 1 for interface problems

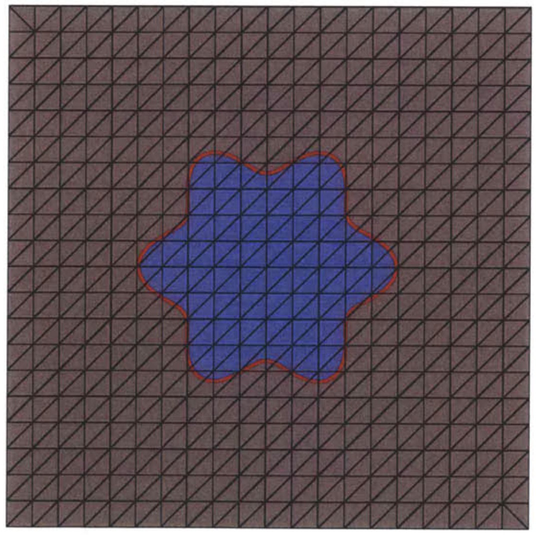


Figure 6-6: Example background mesh for Example 1 for interface problems

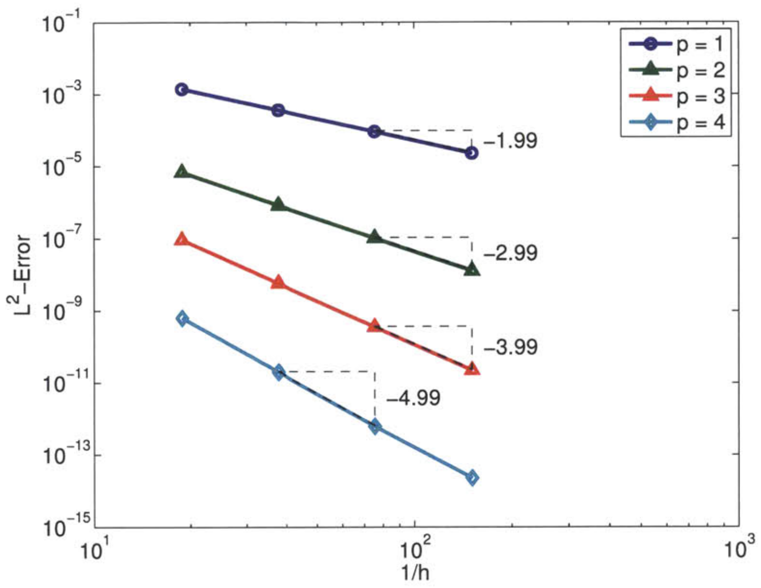


Figure 6-7: L^2-error convergence for Example 1 for interface problems (h defined as $N_{\text{elem}}^{-1/2}$)

16k. The error convergence is shown in Figure 6-9, where we see that the method achieves output superconvergence rate of h^{2p} or DOF^{-p} for both $p = 1$ and $p = 3$. The "DOF-optimal" meshes for $p = 1$ are essentially uniform, and those for $p = 3$ are uniform in $\Omega^{(1)}$ and very coarsen in $\Omega^{(2)}$ because the true solution in $\Omega^{(2)}$ is quadratic and can be exactly captured. Examples of these meshes are shown in Figure 6-10.

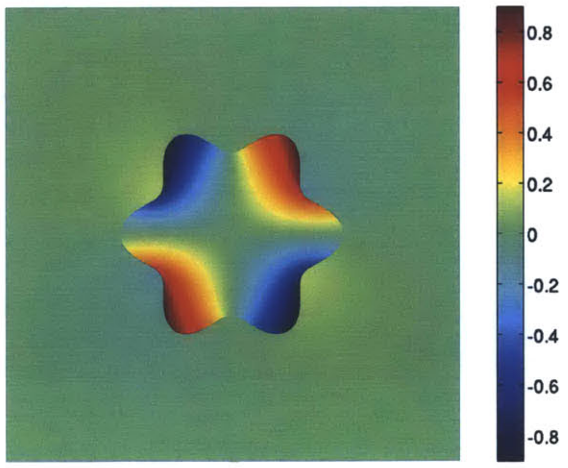


Figure 6-8: Adjoint solution for Example 2 for interface problems

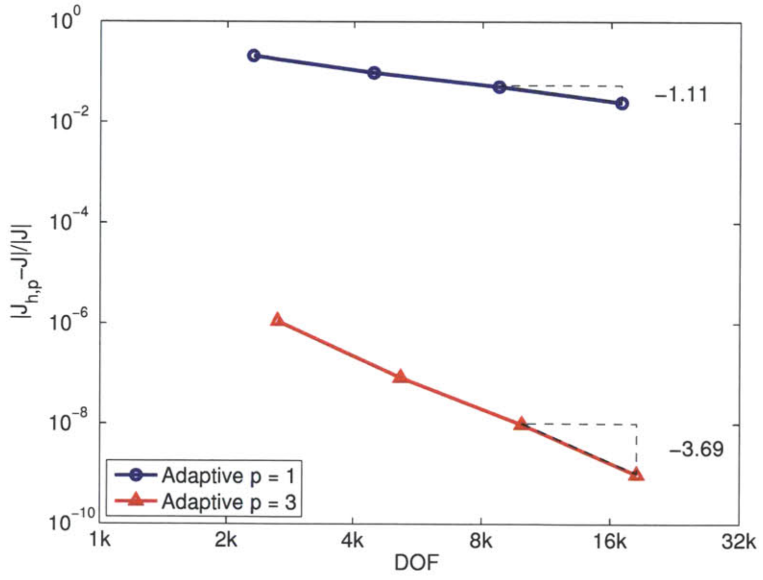
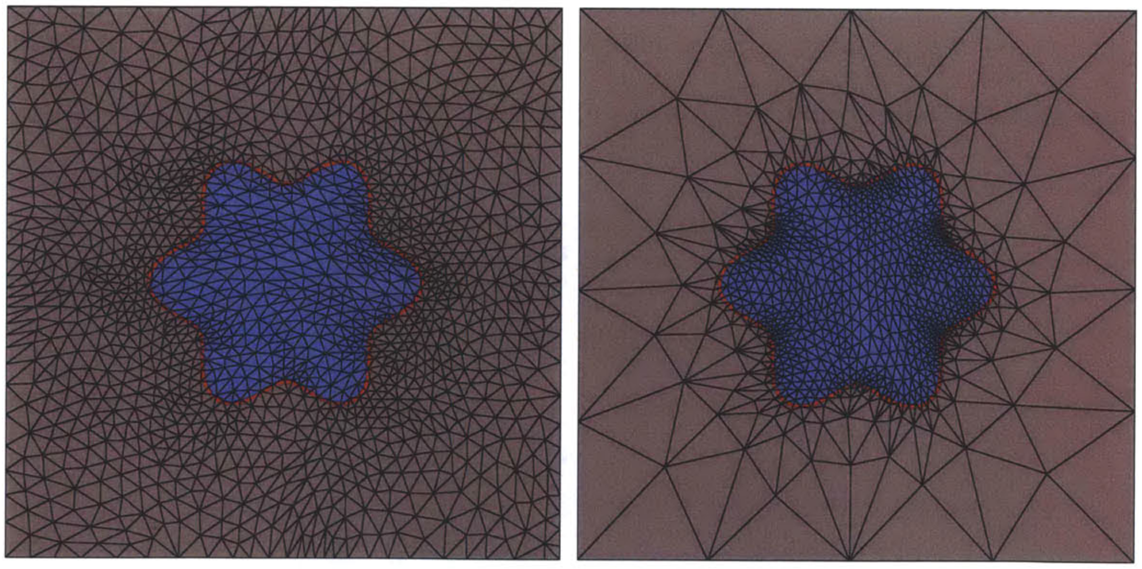


Figure 6-9: Error convergence for Example 2 for interface problems



(a) $p = 1$, DOF $= 8k$ (b) $p = 3$, DOF $= 16k$

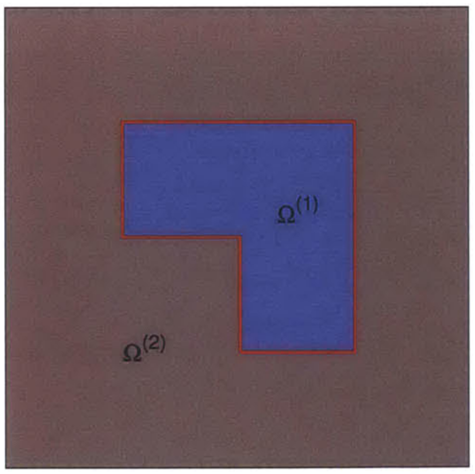
Figure 6-10: Adapted cut-cell meshes for Example 2 for interface problems

Example 3: Geometry-Induced Singularity

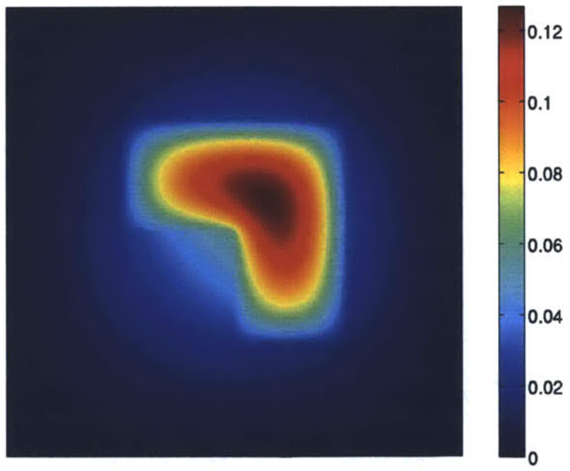
This section presents a case where the interface shape has corners, as shown in Figure 6-11(a). The computational domain is $[-0.5, 0.5] \times [-0.5, 0.5]$, and the interface defines an L-shaped domain with top-left corner at $[-0.25, 0.25]$, bottom-right corner at $[0.25, -0.25]$, and center at $[0, 0]$. The diffusivities are given by $\kappa^{(1)} = 0.1$ and $\kappa^{(2)} = 1$. Homogeneous Dirichlet boundary condition and homogeneous interface condition ($a = b = 0$ in Eq. (6.2)) are imposed, and the source function is

$$f = \begin{cases} 1 & \text{on } \Omega^{(1)} \\ 0 & \text{on } \Omega^{(2)} \end{cases}.$$

The solution is shown in Figure 6-11(b). For this case, the output of interest is the volume term in Eq. (6.22), i.e. $g_{\partial\Omega} = 0$ and $g_\Sigma = 0$, and the integral weight $g_\Omega = f$, which represents an interest only in $\Omega^{(1)}$. For such an output, the adjoint solution is the same as the primal solution shown in Figure 6-11(b).



(a) Computational domain

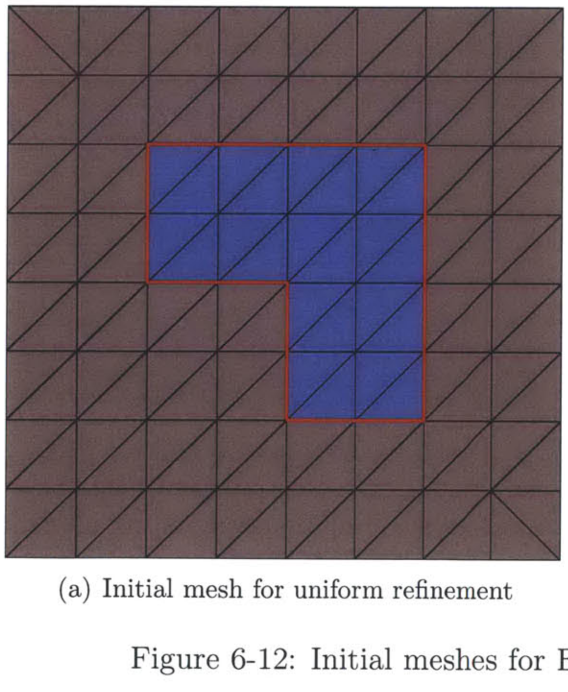


(b) Primal solution

Figure 6-11: Computational domain and primal solution of Example 3 for interface problems

This case is first solved on a sequence of uniformly refined structured meshes that are interface-conforming. The coarsest of these is shown in Figure 6-12(a). Figure 6-13 shows in dashed lines the error convergence on these uniformly refined meshes, where the convergence rate for $p = 3$ is limited by the corner singularities. Note we also tried to solve on a sequence of uniformly refined structured meshes that are not interface-conforming. On such a sequence of meshes, the cut cell shapes on the interface are different from one mesh to the next, and so the error does not converge at the expected rate and hence is not shown. We will see that this "noise" due to arbitrary cut shapes is not apparent on adaptive meshes.

The adaptive scheme is applied for both p's at different DOFs, starting from a uniform unstructured mesh in Figure 6-12(b). For each DOF, 15 adaptation iterations were carried out, and the average of the 5 last iterations at each DOF is shown in Figure 6-13 as solid lines, where output superconvergence of h^{2p} or DOF^{-p} is observed for both p's. Further, the error of each of the 5 last iterations is also plotted, and it is clear that the "noise" caused by arbitrary cut shapes on these adaptive meshes is negligible. Figure 6-14 shows examples of the "DOF-optimal" meshes for $p = 1$ and $p = 3$, where smaller elements are observed at all the corners, and the mesh grading around corners is much stronger for $p = 3$.


(a) Initial mesh for uniform refinement

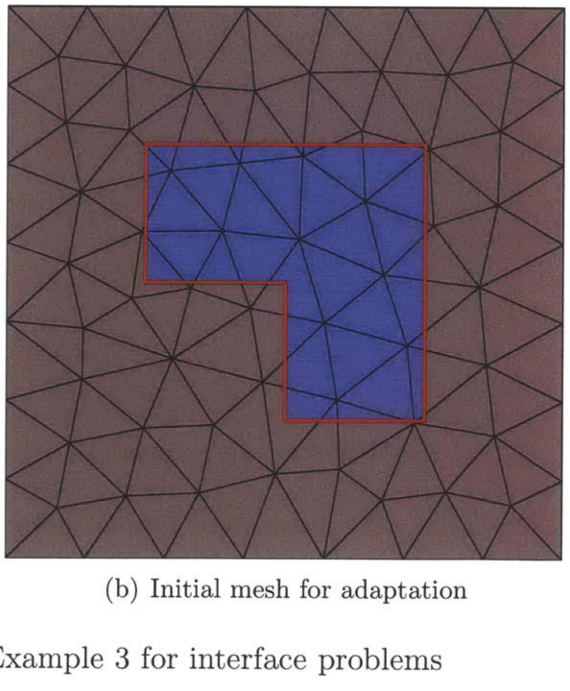

(b) Initial mesh for adaptation

Figure 6-12: Initial meshes for Example 3 for interface problems

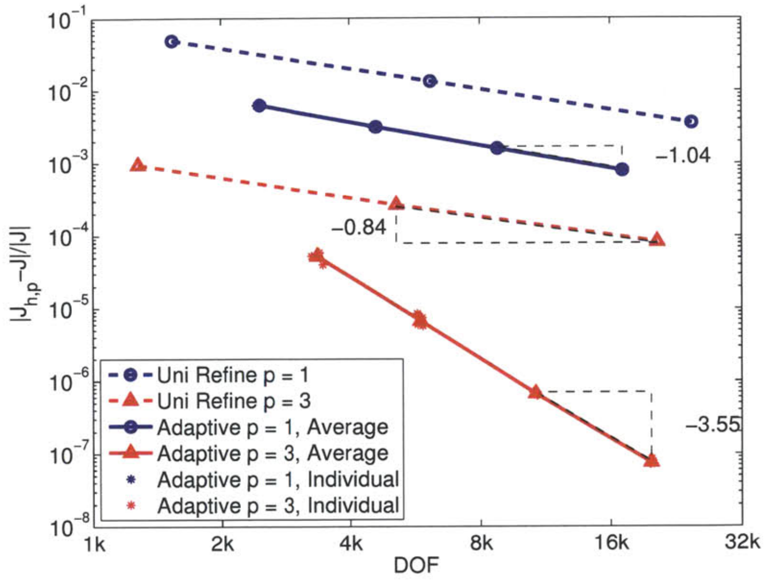
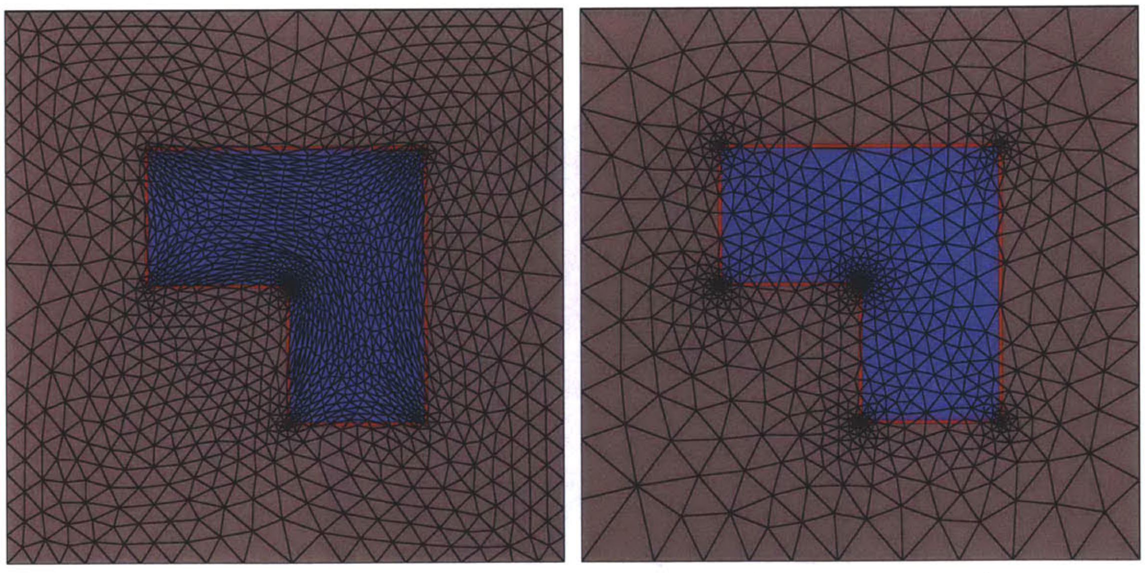


Figure 6-13: Error convergence for Example 3 for interface problems



(a) $p = 1$, DOF $= 8k$ (b) $p = 3$, DOF $= 16k$

Figure 6-14: Adapted cut-cell meshes for Example 3 for interface problems

Example 4: Anisotropic Solution

In this example, we introduce a reaction term into our model PDE:

$$-\nabla \cdot (\kappa \nabla u) + u = f \quad \text{in } \Omega^{(1)} \cup \Omega^{(2)},$$

so that the solution has an anisotropic layer along the interface Σ. The computational domain and the interface are the same as in Example 3, and homogeneous Dirichlet boundary condition and homogeneous interface condition are imposed. The diffusivities are $\kappa^{(1)} = 10^{-4}$ and $\kappa^{(2)} = 1$, and the source function is $f = 1$ in $\Omega^{(1)} \cup \Omega^{(2)}$. Figure 6-15 shows the solution, where a boundary layer exists along the interface.

For this case, the output is the volume integral defined in Eq. (6.22), i.e. $g_{\partial\Omega} = 0$ and $g_{\Sigma} = 0$, and the integral weight $g_{\Omega} = f$ so that the adjoint solution is the same as the primal solution. The adaptation scheme is applied for $p = 1$ and $p = 3$ at different DOFs, starting from the mesh in Figure 6-12(b). Figure 6-16 shows the error convergence, where output superconvergence is observed for both p's. Figure 6-17 shows the "DOF-optimal" meshes for $p = 1$ and $p = 3$, where we observe the adaptive scheme employs strongly anisotropic elements along the interface.

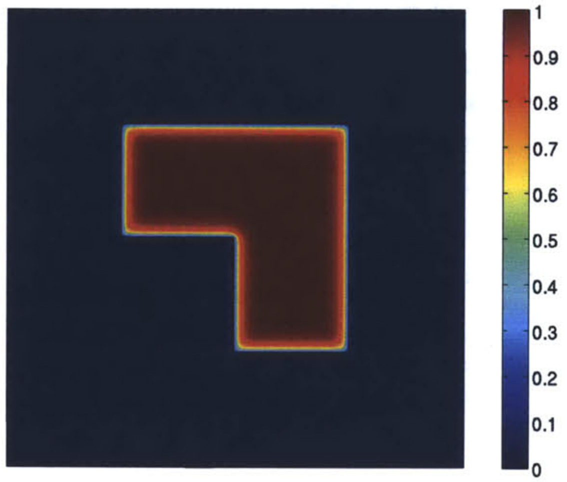


Figure 6-15: Primal solution for Example 4 for interface problems

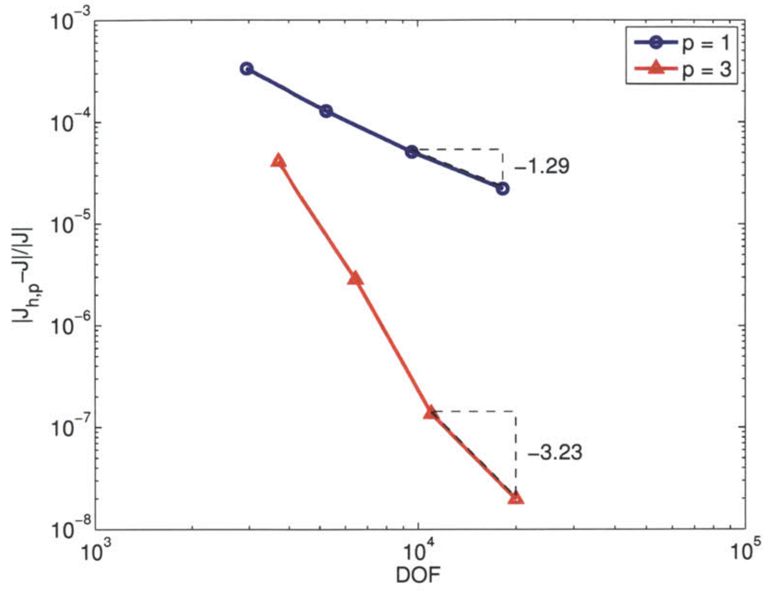
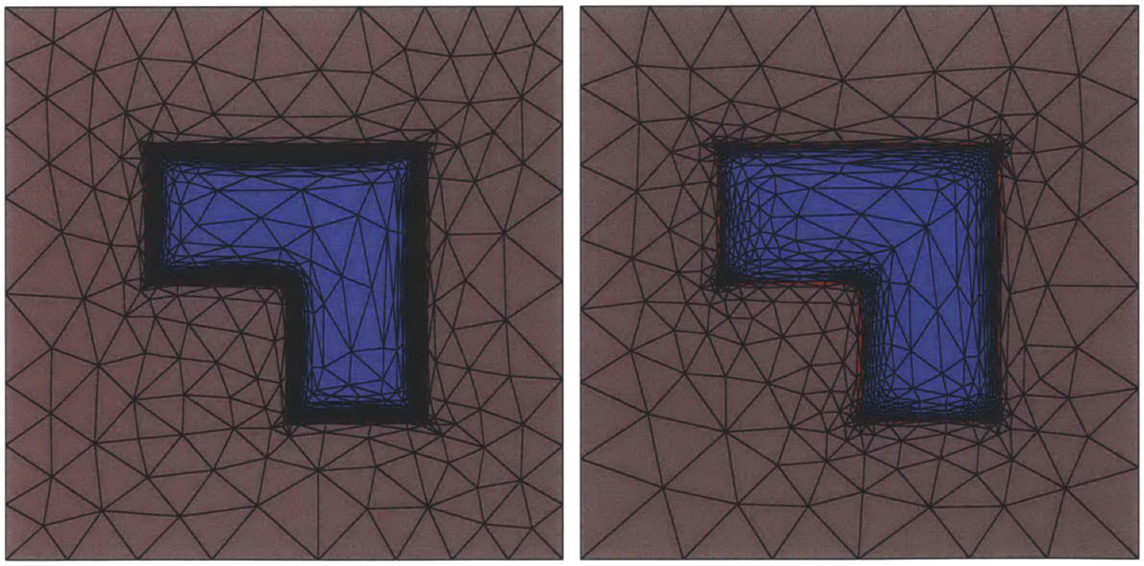


Figure 6-16: Error convergence for Example 4 for interface problems



(a) $p = 1$, DOF $= 8k$ (b) $p = 3$, DOF $= 16k$

Figure 6-17: Adapted cut-cell meshes for Example 4 for interface problems

Chapter 7

Multi-Physics Problems

In this chapter, we extend the framework developed in Chapter 6 to handle multi-physics problems, which are interface problems governed by different PDEs across the interfaces. Section 7.1 derives the DG method for interface problems with general systems of (linear) elliptic equations, which have different parameters and different number of unknown states across the interfaces. Section 7.2 extends the method to conjugate heat transfer (CHT) problems. In Section 7.3, we demonstrate our adaptive, cut-cell framework with the DG discretization through a CHT problem.

7.1 DG for Interface Problems: Systems of Elliptic Equations

In this section, let the computational domain consist of only two sub-domains for simplicity, $\overline{\Omega} = \overline{\Omega}^{(1)} \cup \overline{\Omega}^{(2)}$, and the sub-domains are separated by an interface Σ. On each $\Omega^{(i)}$, the solution $u^{(i)} \in [H^1(\Omega^{(i)})]^{m^{(i)}}$ satisfies

$$-\nabla \cdot \left(\mathcal{A}^{(i)} \nabla u^{(i)}\right) = f^{(i)}, \quad u^{(i)} = 0 \text{ on } \partial\Omega^{(i)} \backslash \Sigma,$$

where $\mathcal{A}^{(i)}$ is the diffusivity tensor, $m^{(i)}$ is the number of unknown states for $u^{(i)}$, and we allow $m^{(1)} \neq m^{(2)}$. Without loss of generality, we assume $m^{(1)} > 1$ and $m^{(2)} = 1$ in this section, and we define $m \equiv m^{(1)}$ for notation simplicity. Assuming $\mathcal{A}^{(i)}$ has a full rank, we impose the following interface conditions on Σ:

$$g(u^{(1)}) = u^{(2)}, \tag{7.1}$$

$$h(\mathcal{A}^{(1)}\nabla u^{(1)} \cdot \hat{\mathbf{n}}^{(1)}) = -\mathcal{A}^{(2)}\nabla u^{(2)} \cdot \hat{\mathbf{n}}^{(2)}, \tag{7.2}$$

$$B(u^{(1)}) = 0, \tag{7.3}$$

where the conditions are assumed to be linear in this section: $g(u^{(1)}) \equiv \mathbf{g}^T u^{(1)}$, $h(\mathcal{A}^{(1)}\nabla u^{(1)} \cdot \hat{\mathbf{n}}^{(1)}) \equiv \mathbf{h}^T(\mathcal{A}^{(1)}\nabla u^{(1)} \cdot \hat{\mathbf{n}}^{(1)})$, and $B(u^{(1)}) = \mathbf{B}^T u^{(1)}$. Conditions (7.1) and (7.2) enforce solution continuity and flux continuity, respectively. Condition (7.3) is for the full specification of $u^{(1)}$ on Σ, where $\mathbf{B} \in \mathbb{R}^{m\times(m-1)}$ has a rank of $m-1$, and each column of $\mathbf{B}$ is linearly independent of $\mathbf{g} \in \mathbb{R}^m$ and $\mathbf{h} \in \mathbb{R}^m$.

For the DG discretization of this problem, the notations and derivations closely follow those in Section 6.2. Recall the notations for different sets of faces: $\Gamma_A^{(i)} \equiv \bigcup_{K\subset\Omega^{(i)}} \partial K$, $\Gamma^{(i)} \equiv \Gamma_A^{(i)}\backslash\Sigma$, and $\Gamma_I^{(i)} \equiv \Gamma_A^{(i)}\backslash\partial\Omega^{(i)}$. Denote the space:

$$V_{h,p}^{(i)} \equiv \{v \in [L^2(\Omega^{(i)})]^{m^{(i)}} : v|_K \in [\mathcal{P}^p(K)]^{m^{(i)}}, \quad \forall K \subset \Omega^{(i)}\}.$$

Then the primal formulation on each $\Omega^{(i)}$, $i = 1, 2$, is identical to (6.8), which is restated here:

$$\begin{aligned}\int_{\Omega^{(i)}} \nabla u_h^{(i)} \cdot \mathcal{A}^{(i)}\nabla v d\Omega &+ \int_{\Gamma^{(i)}} \left(\llbracket \hat{u}^{(i)} - u_h^{(i)} \rrbracket \cdot \{\mathcal{A}^{(i)}\nabla v\} - \{\widehat{\mathcal{A}\sigma}^{(i)}\} \cdot \llbracket v \rrbracket\right) ds \\ &+ \int_{\Gamma_I^{(i)}} \left(\{\hat{u}^{(i)} - u_h^{(i)}\}\llbracket \mathcal{A}^{(i)}\nabla v\rrbracket - \llbracket\widehat{\mathcal{A}\sigma}^{(i)}\rrbracket\{v\}\right) ds \\ &+ \int_{\Sigma} \left((\hat{u}^{(i)} - u_h^{(i)})\mathcal{A}^{(i)}\nabla v - \widehat{\mathcal{A}\sigma}^{(i)} v\right) \cdot \hat{\mathbf{n}}^{(i)} ds = \int_{\Omega^{(i)}} f^{(i)} v d\Omega, \quad \forall v \in V_{h,p}^{(i)}.\end{aligned} \tag{7.4}$$

Note Eq. (7.4) on each sub-domain $\Omega^{(i)}$ has an interface term $\int_\Sigma (\cdot) ds$. This term involves the unknown $u^{(i)}$ evaluated on the interface, which can represent a different physical quantity for a different i. We then define the lifting operators based on (6.7), which is restated:

$$\int_{\Omega^{(i)}} \sigma_h^{(i)} \cdot \tau d\Omega = \int_{\Omega^{(i)}} \nabla u_h^{(i)} \cdot \tau d\Omega + \int_{\Gamma^{(i)}} [\![\hat{u}^{(i)} - u_h^{(i)}]\!] \cdot \{\tau\} ds + \int_{\Gamma_I^{(i)}} \{\hat{u}^{(i)} - u_h^{(i)}\} [\![\tau]\!] ds \\ + \int_{\Sigma} (\hat{u}^{(i)} - u_h^{(i)}) \tau \cdot \hat{\mathbf{n}}^{(i)} ds, \quad \forall \tau \in \left[V_{h,p}^{(i)}\right]^d. \tag{7.5}$$

More specifically, we define $r^{(i)}(\cdot)$, $l^{(i)}(\cdot)$, and $r_{\Sigma}^{(i)}(\cdot)$ by

$$\int_{\Omega^{(i)}} r^{(i)}(\phi) \cdot \tau d\Omega = -\int_{\Gamma^{(i)}} \phi \cdot \{\tau\} ds, \qquad \int_{\Omega^{(i)}} l^{(i)}(q) \cdot \tau d\Omega = -\int_{\Gamma_I^{(i)}} q [\![\tau]\!] ds, \\ \int_{\Omega^{(i)}} r_{\Sigma}^{(i)}(\phi) \cdot \tau d\Omega = -\int_{\Sigma} \phi \tau \cdot \hat{\mathbf{n}}^{(i)} ds \quad \forall \tau \in [V_{h,p}^{(i)}]^d,$$

and Eq. (7.5) leads to

$$\sigma_h^{(i)} = \nabla u_h^{(i)} - r^{(i)}([\![\hat{u} - u_h]\!]) - l^{(i)}(\{\hat{u} - u_h\}) - r_{\Sigma}^{(i)}(\hat{u}^{(i)} - u_h^{(i)}). \tag{7.6}$$

As in Section 6.2.3, we require the numerical fluxes on Σ to satisfy the interface conditions for primal consistency:

$$\mathbf{g}^T \hat{u}^{(1)} = \hat{u}^{(2)}, \quad \mathbf{h}^T(\widehat{\mathcal{A}\sigma}^{(1)} \cdot \hat{\mathbf{n}}^{(1)}) = -\widehat{\mathcal{A}\sigma}^{(2)} \cdot \hat{\mathbf{n}}^{(2)}, \quad \text{and } \mathbf{B}^T \hat{u}^{(1)} = 0,$$

based on which we have the following choices. The numerical fluxes $\hat{u}^{(i)}$, $i = 1, 2$, are defined by solving the linear system

$$\begin{cases} \hat{u}^{(2)} = \mathbf{g}^T \hat{u}^{(1)} = 0.5(\mathbf{g}^T u_h^{(1)} + u_h^{(2)}) \\ \mathbf{B}^T \hat{u}^{(1)} = 0 \end{cases}, \tag{7.7}$$

and $\widehat{A\sigma}^{(i)} \cdot \hat{\mathbf{n}}^{(i)}$, $i = 1, 2$, are defined from

$$\begin{cases} -\widehat{\mathcal{A}\sigma}^{(2)} \cdot \hat{\mathbf{n}}^{(2)} = \mathbf{h}^T(\widehat{\mathcal{A}\sigma}^{(1)} \cdot \hat{\mathbf{n}}^{(1)}) = 0.5(\mathbf{h}^T(\mathcal{A}^{(1)}\sigma_h^{(1)} \cdot \hat{\mathbf{n}}^{(1)}) + \mathcal{A}^{(2)}\sigma_h^{(2)} \cdot \hat{\mathbf{n}}^{(1)}) \\ \mathbf{B}^T(\widehat{\mathcal{A}\sigma}^{(1)} \cdot \hat{\mathbf{n}}^{(1)}) = \mathbf{B}^T(\mathcal{A}^{(1)}\sigma_h^{(1)} \cdot \hat{\mathbf{n}}^{(1)}) \end{cases}, \tag{7.8}$$

where $\sigma_h^{(i)}$ on a face $e \in \Sigma$ is defined by

$$\sigma_h^{(i)} = \nabla u_h^{(i)} - \eta_e r_\Sigma^{e^{(i)}}(\hat{u}^{(i)} - u_h^{(i)}). \tag{7.9}$$

The local lifting operator $r_\Sigma^{e^{(i)}}(\cdot)$ is defined by:

$$\int_{\Omega_h^{e^{(i)}}} r_\Sigma^{e^{(i)}}(\phi) \cdot \tau d\Omega = - \int_e \phi\tau \cdot \hat{\mathbf{n}}^{(i)} ds, \quad \forall \tau \in [V_{h,p}^{(i)}]^d,$$

where $\Omega_h^{e^{(i)}}$ is the element in $\Omega^{(i)}$ neighboring the face e. Note the linear systems $[\mathbf{g}^T; \mathbf{B}^T]$ and $[\mathbf{h}^T; \mathbf{B}^T]$ are invertible as each column of $\mathbf{B}$ is linearly independent of $\mathbf{g}$ and $\mathbf{h}$.

7.2 DG for Interface Problems: Conjugate Heat Transfer

Conjugate heat transfer (CHT) refers to the process of thermal interaction between heat conduction on a solid body and heat convection in an adjacent fluid. Accurate prediction of temperature and/or heat flux distribution in such a process is important in a wide range of applications, for example, turbine blade cooling, aerodynamic heating for re-entry vehicles, and aircraft de-icing. This section applies the DG formulation developed in Section 7.1 to the CHT problems (as a monolithic approach).

As in Section 7.1, we assume two sub-domains in the computational domain, $\overline{\Omega} = \overline{\Omega}^{(1)} \cup \overline{\Omega}^{(2)}$. The sub-domain $\Omega^{(1)}$ is governed by Navier-Stokes equations with unknown $u^{(1)} = [\rho, \rho\mathbf{u}, \rho E]^T$, and $\Omega^{(2)}$ is governed by heat equation with unknown $u^{(2)} = T$. Both equations are described in Appendix E, and a description of the viscosity tensor $\mathcal{A}$ for Navier-Stokes equations can be found in [78]. On the interface Σ, we impose temperature continuity, heat flux continuity, and no-slip condition. More specifically, the general interface conditions (7.1) through (7.3) become

$$T(u^{(1)}) = u^{(2)}, \tag{7.10}$$

$$\mathcal{F}^{\text{heat}}(u^{(1)}) \cdot \hat{\mathbf{n}}^{(1)} = -\mathcal{F}^{\text{heat}}(u^{(2)}) \cdot \hat{\mathbf{n}}^{(2)} \tag{7.11}$$

$$\mathbf{u} = \mathbf{0}, \tag{7.12}$$

where $T(\cdot)$ is the temperature functional, $\mathcal{F}^{\text{heat}} \equiv \kappa_T \nabla T$ is the heat flux functional, and κ_T is the thermal conductivity. Note the main difference from (7.1) through (7.3) is that the viscosity tensor $\mathcal{A}$ for Navier-Stokes equations is rank deficient. More specifically, the viscous flux for these equations contains no mass diffusion, and the conditions (7.10) to (7.12) alone do not fully specify $u^{(1)}$ on Σ.

The DG discretization for this problem has the same bilinear form as given in Eq. (7.4), and the choice of numerical fluxes follows (7.7) and (7.8). Let $\{T\}$ be the average temperature on the interface Σ, $\{T\} \equiv 0.5(T_h^{(1)} + T_h^{(2)})$. Then $\hat{u}^{(i)}$, $i = 1, 2$, are defined by

$$\hat{u}^{(1)} = \begin{bmatrix} \rho_h^{(1)} \\ \mathbf{0} \\ \rho_h^{(1)} \{T\} \frac{R}{\gamma - 1} \end{bmatrix}, \quad \text{and } \hat{u}^{(2)} = \{T\}.$$

The fluxes $\widehat{\mathcal{A}\sigma}^{(i)} \cdot \hat{\mathbf{n}}^{(i)}$, $i = 1, 2$, are defined by

$$\widehat{\mathcal{A}\sigma}^{(1)} \cdot \hat{\mathbf{n}}^{(1)} = \left\{ \begin{array}{c} (\mathcal{A}^{(1)} \sigma_h^{(1)} \cdot \hat{\mathbf{n}}^{(1)})_{\text{mass}} \\ (\mathcal{A}^{(1)} \sigma_h^{(1)} \cdot \hat{\mathbf{n}}^{(1)})_{\text{momentum}} \\ 0.5((\mathcal{A}^{(1)} \sigma_h^{(1)} \cdot \hat{\mathbf{n}}^{(1)})_{\text{energy}} + \kappa_T^{(2)} \sigma_h^{(2)} \cdot \hat{\mathbf{n}}^{(1)}) \end{array} \right. ,$$

$$\text{and } \widehat{\mathcal{A}\sigma}^{(2)} \cdot \hat{\mathbf{n}}^{(2)} = 0.5((\mathcal{A}^{(1)} \sigma_h^{(1)} \cdot \hat{\mathbf{n}}^{(2)})_{\text{energy}} + \kappa_T^{(2)} \sigma_h^{(2)} \cdot \hat{\mathbf{n}}^{(2)}),$$

where the viscosity matrix $\mathcal{A}^{(1)}$ is evaluated at $\hat{u}^{(1)}$, and σ_h is the lifted viscous flux defined in (7.9). Note that at $\hat{u}^{(1)}$, which satisfies the no-slip condition, the term $(\mathcal{A}^{(1)} \sigma_h^{(1)} \cdot \hat{\mathbf{n}}^{(1)})_{\text{energy}}$ is the same as the heat flux, $\kappa_T^{(1)} \sigma_h^{(1)} \cdot \hat{\mathbf{n}}^{(1)}$.

We also need to define the inviscid numerical fluxes $\mathcal{H}^{(i)}$ on the interface Σ. From the side of $\Omega^{(1)}$, $\mathcal{H}^{(1)}$ is computed using $\hat{u}^{(1)}$, and has only the pressure contribution because $\hat{u}^{(1)}$ satisfies the no-slip condition. From the side of $\Omega^{(2)}$ governed by heat equation, the inviscid flux $\mathcal{H}^{(2)}$ is zero.

7.3 Results

As a proof of concept, we demonstrate our solution strategy for conjugate heat transfer in a laminar, external flow, where the geometry is a RAE2822 airfoil with cooling chambers, as shown in Figure 7-1. The ratio of thermal conductivities at room temperature is $\kappa^{(1)}/\kappa^{(2)} = 10^{-3}$. The conductivity $\kappa^{(1)}$ relates to viscosity through Prandtl number, and the viscosity changes with temperature according to Sutherland's law; see Appendix E for more details. The conductivity $\kappa^{(2)}$ is assumed to be constant. The inflow conditions are $M_\infty = 0.3$, $\alpha = 2.31°$, and $Re_c = 10^4$. Cooling is imposed on the wall of the cooling chambers through a Robin boundary condition:

$$-\kappa\nabla T = h(T - T_c),$$

where h is the convective heat transfer coefficient, and T_c is the cooling temperature. In this work, $h = \kappa^{(2)}/c$, and $T_c = 0.5T_\infty$, where c is the airfoil chord length. Figure 7-2 shows the temperature distribution for this case.

The output of interest is the temperature on the airfoil, $\int_{\text{airfoil}} T d\Omega$, and the adaptive scheme is applied for $p = 1$ and $p = 3$ at three different DOFs. The error convergence is shown in Figure 7-3, where we see that the method achieves output superconvergence rate of h^{2p} or DOF^{-p} for both $p = 1$ and $p = 3$. The "DOF-optimal" meshes for $p = 1$ and $p = 3$ are shown in Figure 7-4, where the adaptation automatically adjusts the mesh on both sides of the interface for an accurate prediction of the output. In particular, anisotropic elements are employed for the boundary layer, and isotropic elements inside the airfoil. Also, a mesh grading is observed on the corners of the cooling chambers, especially for the adapted meshes from the $p = 3$ discretization.

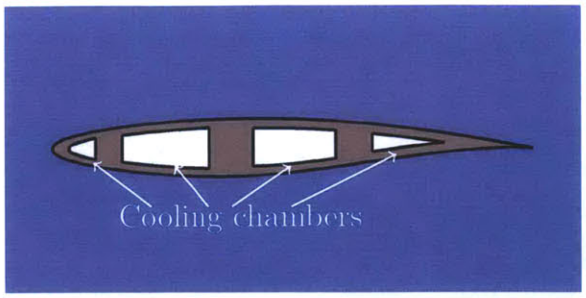
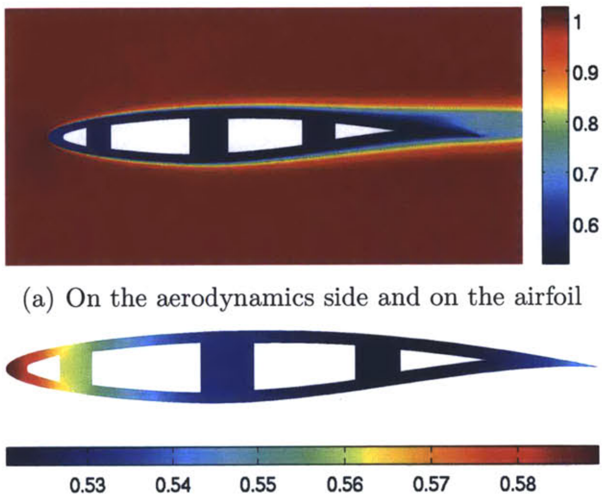


Figure 7-1: RAE2822 airfoil with cooling chambers



(a) On the aerodynamics side and on the airfoil

(b) On the airfoil only

Figure 7-2: Temperature distribution, T/T_∞

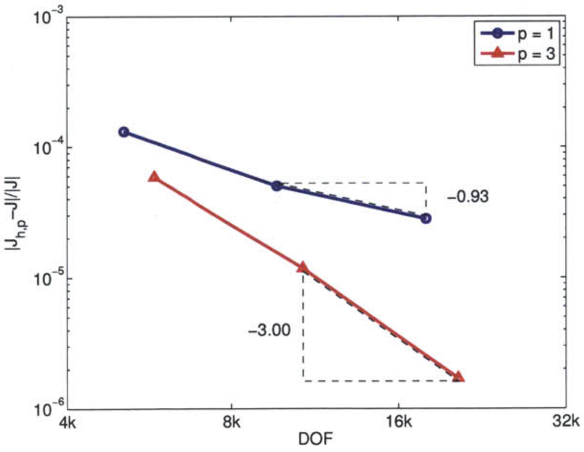
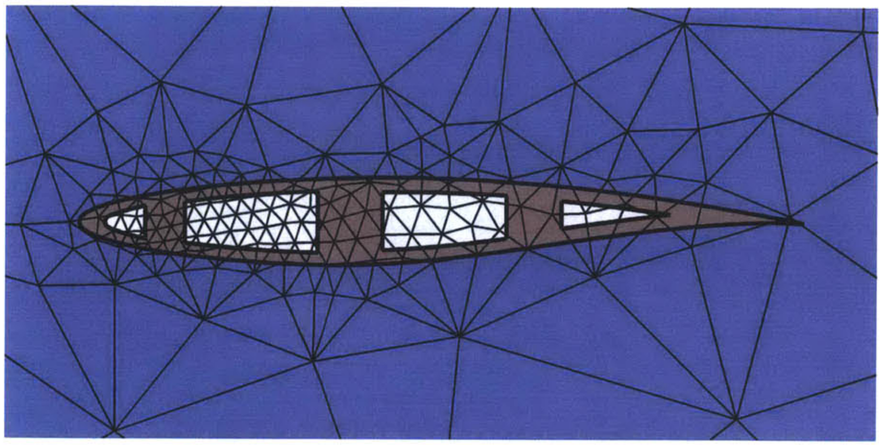
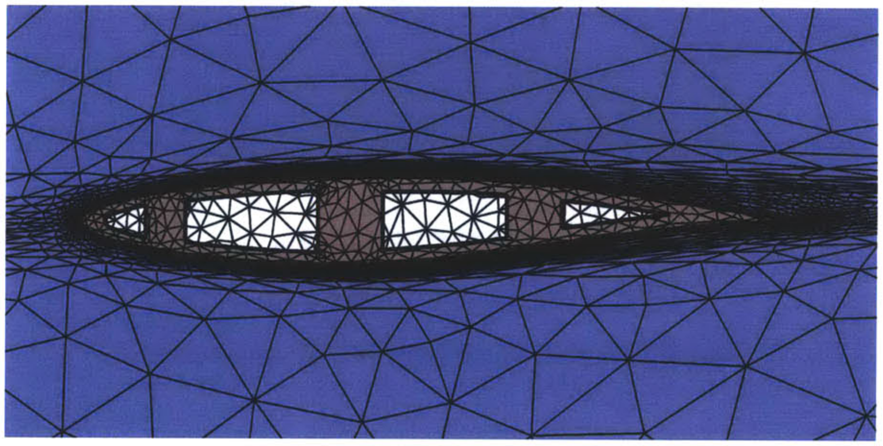


Figure 7-3: Error convergence for the CHT case


(a) Initial mesh


(b) Adapted mesh, $p = 1$, DOF $= 8k$

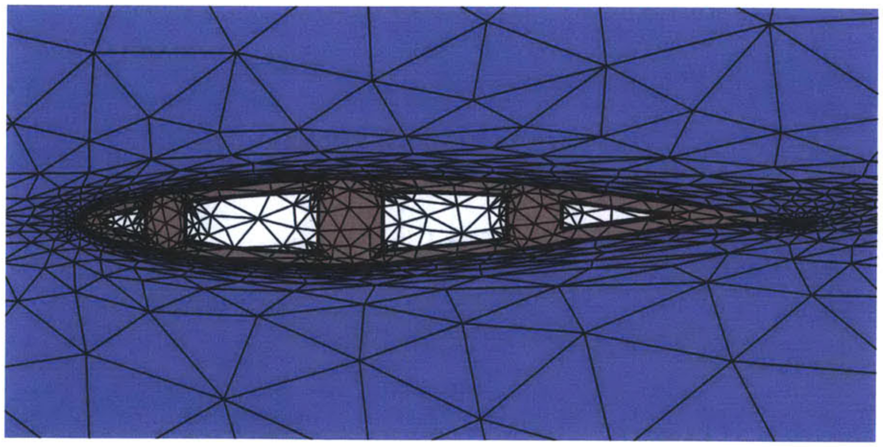

(c) Adapted mesh, $p = 3$, DOF $= 16k$

Figure 7-4: Initial and adapted cut-cell meshes for the CHT case

Chapter 8

Conclusions

8.1 Summary and Conclusions

This thesis presents work toward the development of a robust PDE solution framework that provides a reliable output prediction in a fully-automated manner. In particular, the framework consists of a simplex cut-cell technique, a high-order DG discretization, and an anisotropic output-based adaptation. The simplex cut-cell method is based on the work of Fidkowski and Darmofal [50]. We significantly improved the robustness and automation of their original algorithm in three dimensions by tackling two issues: intersection ambiguity due to numerical precision, and poor quadrature quality for cut cells. In addition, we derived a DG method for multi-material and multi-physics problems, and extended the adaptive, cut-cell framework for these problems.

For the three-dimensional cut-cell intersection problem with the embedded geometry represented by quadratic patches, we demonstrated the robustness issues of using standard *double* precision. We then introduced the adaptive precision arithmetic provided by the LEDA library [26], which guarantees intersection correctness but is computationally unaffordable. Various techniques were developed to improve the efficiency of using the adaptive precision arithmetic. With the improvement, the intersection cost represents only a small fraction of the flow solution cost in terms of CPU time. In addition, the intersection algorithm was parallelized by partitioning the background mesh, and almost a linear speedup was observed.

We proposed a high-quality and efficient cut-cell quadrature rule that satisfies a quality measure we defined. More specifically, the quadrature points are selected based on the idea of the magic points [83], which we proved are asymptotically the same as Fekete points, and improve the quality measure. The quadrature weights are computed based on a weighted least-squares problem to minimize certain projection error. Through an aerodynamics problem, we demonstrated the improvement in nonlinear solver robustness using the proposed quadrature rule. In addition, the proposed algorithm does not rely upon any symmetry information or geometry approximation, and does not involve high-order polynomial root-finding.

With the robust intersection algorithm and the high-quality cut-cell quadrature rule, we demonstrated the automation and robustness of the solution framework through a range of aerodynamics problems, including inviscid and laminar flows. For each presented problem, the cut-cell method was applied on 100 to 200 adapted meshes, ranging from a very coarse mesh with the geometry inside almost one background element, to an adapted mesh with flow features resolved for an accurate output prediction. No human intervention was involved in the process from the initial to the final mesh, including the cut-cell intersection procedure and flow solves using DG polynomial degrees of $p = 1$ and $p = 2$.

We then extended our solution framework for scalar elliptic interface problems. We first derived the DG discretization in a unified form for these problems on fitted meshes, and showed that no modification on the DG bilinear form is needed for interface treatment. We then combined the cut-cell technique, so that the mesh generation process becomes completely separate from the interface definition. No assumption was made on the interface shape (other than Lipschitz continuity). We also extended the MOESS adaptation algorithm [124] to handle cut cells for both embedded boundary and interface problems. Through numerical examples, we demonstrated high-order convergence on cut-cell meshes for elliptic interface problems with both smooth and non-smooth interface shapes. A dual-consistent output evaluation was also derived for the developed DG scheme, and output superconvergence was observed.

We then extended the framework to handle multi-physics problems, which are interface problems governed by different PDEs across the interfaces. The DG method was modified to account for more general interface conditions, and non-interface-conforming meshes were used with the cut-cell technique. The framework was demonstrated (as a monolithic approach) through a conjugate heat transfer problem, where mesh element size and shape on each material are adjusted in a fully-automated manner for an accurate output prediction.

8.2 Future Work

During the course of this work, we identified several areas for potential future research as listed below.

Three-dimensional RANS simulations

With the significant improvements to the robustness of the cut-cell algorithm, simulation of three-dimensional RANS equations on cut-cell meshes can be one potential future work. If successful, this would significantly improve automation of the mesh generation process for aerodynamics problems on complex geometries. Two-dimensional RANS simulations on cut-cell meshes have been demonstrated by Modisette [87].

Three-dimensional interface problems

While we only considered two-dimensional interface problems, our proposed framework is extendable to three dimensions. In particular, the DG discretization developed for interface problems and the adaptation framework have no assumption on dimensions. The intersection and quadrature rule generation developed for embedded boundary problems in this thesis are also applicable for interface problems, and the implementation will be considered in future.

Adaptation of quadratic patches

One source of numerical error that was not considered in this thesis is the quadratic-patch approximation of the geometry surface, which is often represented by a CAD model. This error can be more prominent if a higher-fidelity simulation is required, using for example a higher approximation polynomial degree. One possible solution is to identify the impact of this error on the output prediction through an adjoint analysis. Then an automated adjustment of the quadratic-patch resolution can be incorporated in the adaptive framework.

Extension to problems with moving geometries

While the applications in this thesis are mainly for aerodynamics and heat transfer, the proposed PDE solution framework can be extendable to other applications governed by different PDEs. One particular interest is the extension to problems with a moving boundary or interface, for example, the aeroelastic studies based on fluid-structure interactions. While the cut-cell intersection will need to be carried out much more frequently for such problems, a variety of other issues must also be addressed, including adjoint analysis and output-based adaptation for unsteady (and possibly chaotic) systems.

Appendix A

Notes on Cut-Cell Intersection Algorithm

A.1 Sign Computation for Algebraic Numbers

In the library of LEDA [26] (or CORE [44]), every algebraic number has its entire construction history stored as a directed acyclic graph (DAG), whose internal nodes represent arithmetic operations (e.g. $+$) and whose leaf nodes are the input numbers. Figure A-1 shows an example of DAG, for the number $(\sqrt{17}+\sqrt{12})\times(\sqrt{17}-\sqrt{12})-5$. Each internal node is also stored with the accumulated round-off error involved, and a positive number known as separation bound used for sign computation, which will be explained later in this appendix.

Let E represent an arithmetic expression with value f. With finite precision, let the upper bound for round-off error be δ, i.e. $|f-\tilde{f}|<\delta$ where $\tilde{f}$ is the approximated f. The problem we want to solve is the sign of f given the value of $\tilde{f}$ and δ. This can be easily solved when f is not zero: if $|\tilde{f}|>\delta$, then f and $\tilde{f}$ have the same sign; otherwise $|\tilde{f}|\leq\delta$, we then keep refining the precision for $\tilde{f}$ and hence reduce δ until we have $|\tilde{f}|>\delta$. An illustration is shown in Figure A-2. This problem is more complicated when $f=0$, as precision refinement for $\tilde{f}$ will drive both $\tilde{f}$ and δ to zero.

In the case of $f=0$, we then need the concept of separation bound to indicate

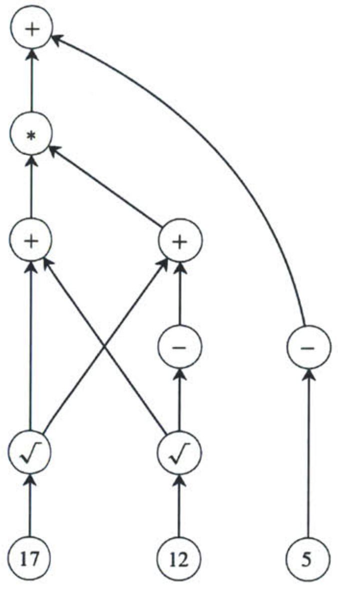
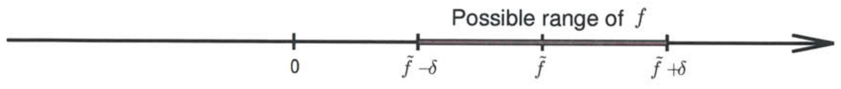
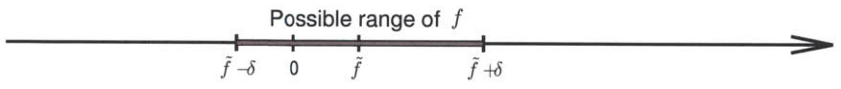


Figure A-1: Example of a directed acyclic graph for $(\sqrt{17}+\sqrt{12}) \times (\sqrt{17}-\sqrt{12}) - 5$ (LEDA Manual)

f is identically zero. A separation bound $sep(E)$ for the expression E with value f is a positive number such that $|f| < sep(E)$ implies $f = 0$. A simple example of separation bound on integer arithmetic is the number 1. For algebraic expressions, separation bounds are always computable; see for example [25]. Therefore, if $|\tilde{f}| \leq \delta$, or equivalently $|f| \leq 2\delta$, we then keep refining the precision for $\tilde{f}$ until we have either $|\tilde{f}| > \delta$ or $2\delta < sep(E)$. In the latter case, we then have $|f| \leq 2\delta < sep(E)$, and then $f = 0$ can be deduced.



(a) $|\tilde{f}| > \delta$, then f and $\tilde{f}$ have the same sign



(b) $|\tilde{f}| \leq \delta$, then need to reduce δ to compute sign(f)

Figure A-2: Sign computation for f given $\tilde{f}$ and δ, where $|f - \tilde{f}| < \delta$

A.2 More on Efficiency Improvement

Various techniques were developed to make the adaptive precision arithmetic affordable for the cut-cell intersection problem. Section 2.4 describes the key concepts while this appendix provides additional discussion.

Background-Edge Intersection

As described in Section 2.3, an intersection problem between a background edge AB and a quadratic patch $\{\mathbf{x}_i\}_{i=1}^{6}$ can be formulated as in Eq. (2.2), which is rewritten here: find X, Y, and t such that

$$\sum_{i=1}^{6} \phi_i(X,Y)\mathbf{x}_i = \mathbf{x}_A + t(\mathbf{x}_B - \mathbf{x}_A) \qquad \text{(A.1)}$$
$$t \in [0,1], \quad X, Y \geq 0, \quad X + Y \leq 1.$$

The details of how this system is solved with the adaptive precision arithmetic are described in Section 2.4.

Another formulation is to represent the background edge as the intersection line between two planes (e.g. two adjoining background faces). Denote their normal vectors by $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$, respectively. Then in the patch reference space, the intersection curve between each plane and the patch is given by

$$S_j(X,Y) \equiv \left(\sum_{i=1}^{6} \phi_i(X,Y)\mathbf{x}_i - \mathbf{x}_A \right) \cdot \hat{\mathbf{n}}_j = 0, \quad j = 1,2. \qquad \text{(A.2)}$$

The intersection point between the background edge and the patch is thus the solution of the system $S_1(X,Y) = 0$ and $S_2(X,Y) = 0$, which represents a conic-conic intersection problem. This formulation is the same as Fidkowski's implementation [49].

In this work, both formulations were implemented using the adaptive precision arithmetic, where the conic-conic intersection problem is solved using the method described in Section 2.4. Figure A-3 shows a test case where point A is fixed, the intersection point P is uniformly sampled on the patch, and B is extended from the line AP such that $|AP| = |BP|$. We then solve the intersection problem between AB and the patch for 100 times using both formulations, and the difference in efficiency is listed in Table A.1 [1]. The formulation from Eq. (A.1) is about two times faster, as the coefficients for the system (A.1) have simpler construction DAG. Note the two formulations have no difference in speed for *double* precision.

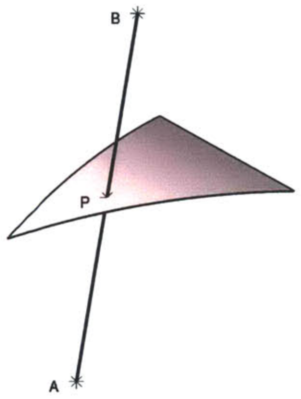


Figure A-3: A test case with random background-edge intersection point

Table A.1: Efficiency of the two formulations for background-edge intersections

Formulation	Wall Clock Time (s)
Eq. (A.1)	0.96
Eq. (A.2)	2.01

Sorting Points Along Conic Sections: Branch Determination

As described in Section B.2, we need to sort a set of points, $\{X_i, Y_i\}_{i=1}^N$, along conic sections. For conics with two branches, i.e. hyperbola and degenerate conics with two lines, we need to first group the points onto each branch. For hyperbola, we first find one of its asymptotes, $aX + bY + c = 0$; see Art. 174 in [113]. Then for each point $\{X_i, Y_i\}$, we determine the branch according to the sign of $aX_i + bY_i + c$.

[1]Wall clock time measured on an Intel Xeon 5570 processor at 2.93GHz. O/S: Ubuntu 11.04.

For degenerate conics with two lines, let the two lines be denoted by $a_1X + b_1Y + c_1 = 0$ and $a_2X + b_2Y + c_2 = 0$. One way to determine which line has $\{X_i, Y_i\}$ is to determine whether $a_1X_i + b_1Y_i + c_1$ or $a_2X_i + b_2Y_i + c_2$ is zero. This leads to severe efficiency issues as we query the sign of an expression that is identically zero and may have a very complex construction DAG involving solutions from conic-conic intersections. One alternative is to define for each $\{X_i, Y_i\}$:

$$\begin{aligned}\Delta_1 &= (a_1X_i + b_1Y_i + c_1) + (a_2X_i + b_2Y_i + c_2),\\ \Delta_2 &= (a_1X_i + b_1Y_i + c_1) - (a_2X_i + b_2Y_i + c_2).\end{aligned}$$

Then Δ_1 and Δ_2 are neither zero, and if they have the same sign, $\{X_i, Y_i\}$ is on the line $a_2X + b_2Y + c_2 = 0$; otherwise, $\{X_i, Y_i\}$ is on $a_1X + b_1Y + c_1 = 0$.

Special Case for Linear Patches

When the geometry surface is planar, the quadratic patches defined by (2.1) reduce to linear, and so does the system (2.2). While we can still use the method described in Section 2.4, the conics in the system (2.5) will be straight lines. As all the coefficients for quadratic terms are identically zero, querying their signs with the adaptive precision arithmetic causes significant slowdown for the intersection algorithm. Therefore, as an input to the intersection algorithm, we explicitly specify a patch is linear if the geometry is planar, and assign all the quadratic terms to zero without computation.

Reuse of Data

In the whole intersection algorithm, we always try to reuse the variables whose precision has been refined. For example, the normal vectors of background faces (or conic coefficients) are present in the construction DAG for many intersection points, and so may have a refined precision. They should be reused when needed during the stage of *oned* construction.

A.3 Classification and Sturm's Sequence for Quartic Equations

Let a quartic equation be defined by

$$f(x) \equiv ax^4 - 4bx^3 + 6cx^2 - 4dx + e = 0. \tag{A.3}$$

Without loss of generality, we assume $a > 0$ in this section. The classification of quartic equations based on the number of real roots can be found in, for example, [46]. Using the notations in [46], we define

$$\begin{aligned} W_1 &= ad - bc, & \Delta_1 &= A^3 - 27B^2, & T_1 &= -W_3\Delta_2 - 3W_1^2 + 9\Delta_2\Delta_3, \\ W_2 &= be - cd, & \Delta_2 &= b^2 - ac, & T_2 &= AW_1 - 3dB, \\ W_3 &= ae - bd, & \Delta_3 &= c^2 - bd, \end{aligned}$$

where $A = W_3 + 3\Delta_3$ and $B = -dW_1 - e\Delta_2 - c\Delta_3$. Note a small typographical error for T_2 in [46]. All these terms are used in the classification of the quartic equation, and we will see that its Sturm sequence can also be expressed using these terms.

Let $P(x)$ be a polynomial in x. The Sturm sequence of polynomials is defined as

$$\begin{aligned} &P_0(x) = P(x), \qquad P_1(x) = P'(x) \\ &P_n(x) = -\text{rem}(P_{n-2}, P_{n-1}), \quad n \geq 2, \end{aligned}$$

where $\text{rem}(P_{n-2}, P_{n-1})$ denotes the remainder of P_{n-2} upon division by P_{n-1}. The sequence terminates at P_i once it is zero. The Sturm sequence of a general quartic polynomials defined in (A.3), with $\Delta_2 \neq 0$ and $T_1 \neq 0$, can be derived to be

$$\begin{aligned} &P_0 = ax^4 - 4bx^3 + 6cx^2 - 4dx + e, \quad P_1 = 4ax^3 - 12bx^2 + 12cx - 4d, \\ &\quad P_2 = \frac{1}{a}(3\Delta_2 x^2 + 3W_1 x - W_3), \quad P_3 = \frac{4a}{3\Delta_2^2}(T_1 x + T_2), \quad P_4 = \frac{\Delta_1\Delta_2^2}{aT_1^2}. \end{aligned}$$

Because we need only signs of these polynomials, we redefine $\{P_i\}_{i=2}^4$ as

$$P_2 = 3\Delta_2 x^2 + 3W_1 x - W_3, \quad P_3 = T_1 x + T_2, \quad P_4 = \Delta_1.$$

There are several special cases:

- $\Delta_2 = 0$, $W_1 \neq 0$: the Sturm sequence terminates at P_3 with

  $$P_2 = 3W_1 x - W_3, \quad P_3 = -\frac{4a\Delta_1}{27W_1^3},$$

  where the sign of P_3 is the same as $-\Delta_1 W_1$.

- $\Delta_2 = 0$, $W_1 = 0$: the Sturm sequence terminates at P_2, with $P_2 = -W_3$;

- $\Delta_2 \neq 0$, $T_1 = 0$: the Sturm sequence terminates at P_3, with $P_3 = T_2$.

A.4 Nearly-Duplicate Roots for Cubic Equations

For the conic-conic intersection problem described in Section 2.4, a cubic equation is encountered, and only one real root is needed. In LEDA, the root for a cubic equation is represented using the *diamond* operator for algebraic numbers [111], which applies Newton's method for one specified real root (i.e. the smallest, the second smallest or the largest). It is thus critical not to seek a duplicate or nearly-duplicate root due to efficiency concerns. In this section, we develop methods to identify and avoid a duplicate or nearly-duplicate root without solving the equation.

Let a cubic equation be defined by

$$ax^3 + bx^2 + cx + d = 0.$$

As described in [46], the classification of the equation based on the number of real roots uses these terms:

$$\begin{aligned} &\Delta_1 = -\tfrac{1}{3}(W^2 - 4\Delta_2\Delta_3), \qquad &&\Delta_2 = b^2 - 3ac, \qquad &&\Delta_3 = c^2 - 3bd, \\ &W = bc - 9ad, &&P = 2b\Delta_2 - 3aW. \end{aligned} \tag{A.4}$$

In this section, we always assume the cubic equation has three distinct real roots, because we have analytical formula with only algebraic numbers otherwise.

Let a cubic equation (with three distinct real roots) be constructed by

$$(x - x_0)(x - x_0 - \alpha)(x - x_0 - \beta) = 0, \quad \beta > \alpha > 0,$$

which has roots: x_0, $x_0+\alpha$, and $x_0+\beta$. We can show that the discriminant Δ_1 defined in (A.4) is related to the distance among the three roots by

$$\Delta_1 = \alpha^2\beta^2(\beta - \alpha)^2.$$

Further, define $t \equiv \alpha/\beta \in (0, 1)$, then t being close to 0 or 1 indicates the existence of two roots that are close to each other, i.e. two nearly-degenerate real roots. We can show that $\Delta_2 = \alpha^2 + \beta^2 - \alpha\beta$, and so derive the quantity:

$$\Delta \equiv \frac{\Delta_1^{1/3}}{\Delta_2} = \frac{(t^2 - t)^{2/3}}{t^2 - t + 1}.$$

Figure A-4 shows the plot of Δ, and it is clear that Δ can be an indicator for the magnitude of t; if Δ is below some specified threshold, we conclude there are nearly-degenerate roots. Further, the indicator Δ involves only Δ_1 and Δ_2, the signs of which have been determined when classifying the cubic equation.

When there are indeed two nearly-degenerate roots, the third root should be the one we seek for using the *diamond* operator. Thus we need to identify the relative position of the third root, i.e. whether it is the smallest or the largest. We can deduce

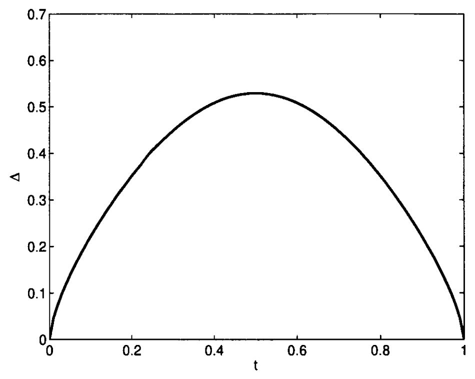


Figure A-4: Detecting nearly-degenerate roots for cubic equations

this information based on the sign of P defined in (A.4), which can be derived to be

$$P = -\beta^2(\alpha + \beta)(t - 2)(2t - 1).$$

When the two nearly-degenerate roots are smaller than the third root (i.e. t is close to 0), we have $P < 0$; and when the two nearly-degenerate roots are larger than the third root (i.e. t is close to 1), we have $P > 0$.

Appendix B

Cut-Cell Mesh Construction

The cut-cell intersection algorithm constructs the topology of the cut mesh from a simplex background mesh and a geometry defined by quadratic patches. The skeleton of the intersection algorithm in this work is similar to Fidkowski's implementation [49]. Changes are made mainly for numerical conditioning concerns and for efficiency improvement to use the adaptive precision arithmetic, as discussed in Chapter 2. This appendix provides the detail for each of the four steps in the intersection algorithm:

1. computation of intersection points, named *zerod* objects, or simply *zerod* for brevity;

2. construction of intersection edges (*oned* objects or simply *oned*) by ordering and connecting the *zerod* objects;

3. construction of intersection faces (*twod* objects or simply *twod*) by connecting the *oned* objects into loops;

4. construction of cut elements (*threed* objects or simply *threed*) by making the *twod* objects into closed volumes.

Note in this appendix, the adaptive precision arithmetic provided by LEDA *real* [26] is used for all calculations unless otherwise stated.

An example of one single tetrahedron intersecting quadratic patches is illustrated in Figure 2-1 in Chapter 2, and is shown here again in Figure B-1. For presentation convenience, we define a background edge (or face) as an edge (or face) of a background element, for example AB in Figure B-1(a) as a background edge, and ACB as a background face. We also define a patch edge as an edge of a (quadratic) patch on the geometry surface.

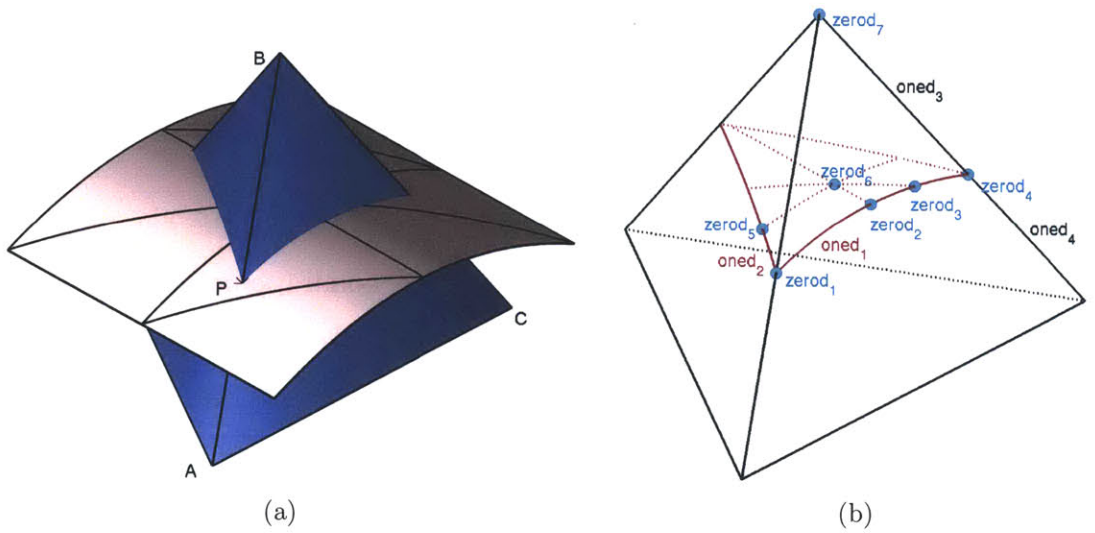


Figure B-1: Example intersection between a background tetrahedron and a quadratic-patch surface

B.1 Construction of Intersection Points (*zerod* Objects)

This section describes the first step of the intersection algorithm: computation of all the intersection points (*zerod* objects). A high-level pseudocode for this step is provided in Algorithm B.1, and details are in the rest of this section. Algorithm B.1 is decomposed into three parts based on the positions of the intersection points:

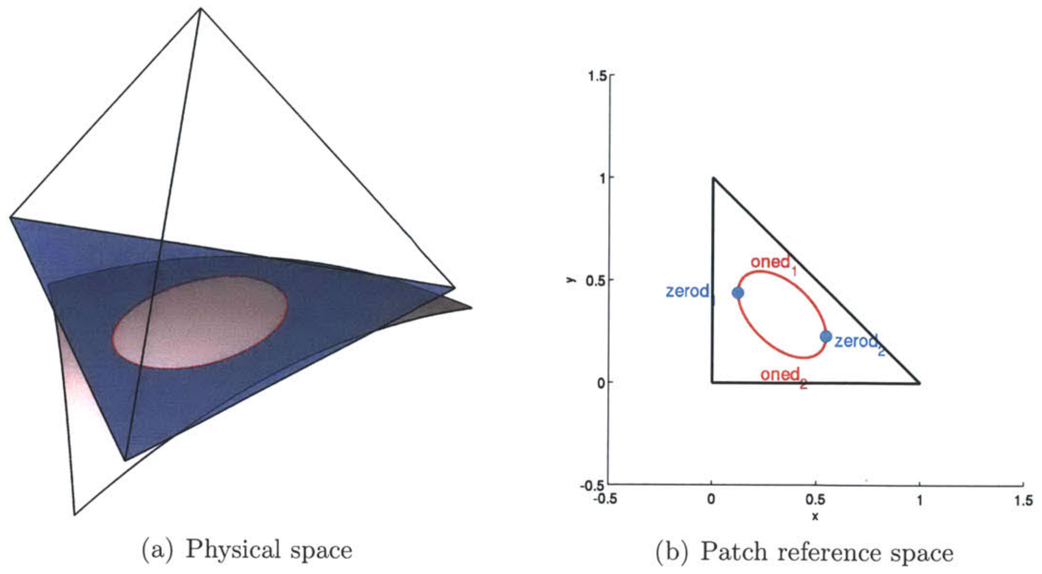
1. Find intersection points between all patch edges and the background mesh.

2. Find intersection points between all background edges and the geometry surface.

3. Detect cases where the intersection curve between a patch and a background face lies completely inside the patch and the face, i.e. no intersection point exists on the patch edge or the background edge. An example is illustrated in Figure B-2(a). For such a case, we store two points on the intersection curve as *zerod* objects.

```
Input: Background grid, quadratic-patch geometry
Output: zerod objects
/* Part 1: patch-edge intersection points                          */
for each patch edge $e_p$ do
    for each background face $f_b$ do
        if intersection detected by bounding-box test then
            Solve the intersection points between $e_p$ and $f_b$ based on Eq. (B.2)
        end
    end
end
/* Part 2: background-edge intersection points                     */
for each background edge $e_b$ do
    for each patch face $f_p$ do
        if intersection detected by bounding-box test then
            Solve the intersection points between $e_b$ and $f_p$ based on Eq. (2.2);
        end
    end
end
/* Part 3: Background-face-patch-face intersection points          */
for each patch face $f_p$ do
    for each background face $f_b$ do
        $\mathcal{C} \leftarrow$ intersection conic between $f_p$ and $f_b$ in the reference space of $f_p$;
        if $\mathcal{C}$ is an ellipse and $\mathcal{C}$ has no intersection point associated then
            Store the two extrema points of $\mathcal{C}$ as zerod objects ;
        end
    end
end
```

Algorithm B.1: Construction of *zerod* objects



(a) Physical space (b) Patch reference space

Figure B-2: Example where the intersection curve (shown as red line) lies completely inside a patch face and a background face

Part 1: Patch-Edge Intersection Points

We first find all the intersection points on patch edges, including those coincident with patch vertices. This is achieved by solving an intersection problem for each pair of a patch edge e_p and a background face f_b. Note these intersection points can also lie on background edges or coincide with background vertices. Examples of the patch-edge intersection points are $zerod_2$ and $zerod_5$ in Figure B-1(b).

For the pair of e_p and f_b, a bounding-box test is first performed using *double* precision. Using the method of separating axis [45], we examine whether f_b has an overlap with the bounding boxes of the neighbor patches of e_p. Note the bounding box of a patch is defined based on the its extrema in each coordinate direction. We then proceed to the intersection problem only if an overlap is detected.

Let the plane containing the background face f_b be defined by its normal vector $\vec{\mathbf{n}}$ and any point on the plane $\mathbf{v}_0$. The patch edge e_p can be shown to be a planar curve mapping from a unit segment, via

$$\mathbf{x}(s) = \sum_{j=1}^{3} \psi_j(s)\mathbf{x}_j, \tag{B.1}$$

where $s \in [0, 1]$ is the coordinate along the patch edge, ψ_j's are the quadratic Lagrange polynomials defined on a unit segment, and $\mathbf{x}_j$'s are the nodes defining the edge. Then the intersection points satisfy

$$\left(\sum_{j=1}^{3} \psi_j(s)\mathbf{x}_j - \mathbf{v}_0 \right) \cdot \vec{\mathbf{n}} = 0, \tag{B.2}$$

which is a quadratic equation in s. We determine whether there are roots in $[0, 1]$ based on the equation coefficients, and then solve the equation if such roots exist. Note for this simple quadratic equation, we can also directly solve for all the roots and then determine the range. The speed difference is not significant except in some rare cases (e.g. a tangent intersection point coincident with patch vertex). We then find the physical coordinates of the intersection points from Eq. (B.1), and determine whether these points lie inside the background face f_b.

There exists a special case where the patch edge e_p lies entirely inside the background face f_b. Such a case is detected when all the coefficients of the quadratic equation (B.2) are zero. For this case, the two vertices of e_p are stored as *zerod* objects.

Part 2: Background-Edge Intersection

We then find all the intersection points on background edges, including those coincident with background vertices. This is achieved by solving an intersection problem for each pair of a background edge e_b and a patch face f_p. Examples of such intersection points are *zerod*$_1$ and *zerod*$_4$ in Figure B-1(b). Note a quadratic patch face can intersect with a line for up to four times; see the work of Peters and Reif, who classified all possible quadratic surfaces [100].

For the pair of e_b and f_p, a bounding-box test is first performed using *double* precision. We examine whether the bounding box of the patch f_p has an overlap with the background edge e_b, and we proceed to the intersection problem only if an overlap is detected. The intersection problem is governed by the polynomial system (2.4), and as described in Section 2.4, we convert the system into a bivariate quadratic system (2.5). We then determine the existence of real roots in the range of $[0,1]^2$, and solve the conic-conic intersection using the method discussed in Section 2.4. Note this formulation is different from Fidkowski's implementation [49], and the efficiency implication of this difference is discussed in Appendix A.2.

Part 3: Background-Face-Patch-Face Intersection

We then detect the case where the intersection curve between a patch and background face lies completely inside the patch and the face, as shown in Figure B-2 for an example. Such a case has an intersection conic that is an ellipse lying completely inside the reference triangle of the patch and has no intersection points (*zerod* objects) associated. To verify whether an ellipse is contained in a triangle, we inspect whether its extrema points (points with maximum and minimum x-coordinate) are inside the triangle. For data storage, we store the two extrema points as *zerod* objects as shown in Figure B-2(b), which will be connected by *oned* objects.

B.2 Construction of Intersection Edges (*oned* Objects)

From the set of *zerod* objects, a set of *oned* objects is then built. For instance, in Figure B-1, $oned_1$ links $zerod_1$ and $zerod_2$. A high-level pseudocode for the construction of *oned* objects is in Algorithm B.2, which consists of three parts separated based on the positions of the *oned* objects. Details of each part of the algorithm are given in the rest of this section.

```
Input: Background grid, quadratic-patch geometry, zerod objects
Output: oned objects
/* Part 1:  Patch-edge oned objects                                   */
for each patch edge $e_p$ do
    Collect all zerod objects on $e_p$;
    Sort these zerod's based on their reference coordinates on $e_p$;
    for each two consecutive zerod objects, $\{zerod_i,\ zerod_{i+1}\}$ do
        Create an oned object, $oned_i$, that links $zerod_i$ and $zerod_{i+1}$ ;
        if $oned_i$ is outside computational domain then
            Remove $oned_i$ from the oned list;
        else
            Find the index of background element (or face) that $oned_i$ lies in;
        end
    end
end
/* Part 2:  Background-edge oned objects                              */
for each background edge $e_b$ do
    Collect all zerod objects on $e_b$;
    Sort these zerod's based on their reference coordinates on $e_b$;
    for each two consecutive zerod objects, $\{zerod_i,\ zerod_{i+1}\}$ do
        Create an oned object, $oned_i$, that links $zerod_i$ and $zerod_{i+1}$ ;
        if $oned_i$ is outside computational domain then
            Remove $oned_i$ from the oned list;
        end
    end
end
/* Part 3:  Patch-face oned objects                                   */
for each patch face $f_p$ do
    for each background face $f_b$ do
        $C \leftarrow$ intersection conic between $f_p$ and $f_b$ in the reference space of $f_p$;
        Collect all zerod objects on $C$;
        Sort these zerod's on $C$ by taking advantage of the convexity of $C$;
        for each two consecutive zerod objects, $\{zerod_i,\ zerod_{i+1}\}$ do
            Create an oned object, $oned_i$, that links $zerod_i$ and $zerod_{i+1}$ ;
            if $oned_i$ is outside the reference triangle of $f_p$
              or $oned_i$ is outside of $f_b$ in the physical space then
                Remove $oned_i$ from the oned list;
            end
        end
    end
end
```

Algorithm B.2: Construction of *oned* objects

Part 1: Patch-Edge *oned* Objects

We first construct the *oned* objects on patch edges. For each patch edge, we sort all the associated *zerod* objects based on their reference coordinates on the edge, and each two consecutive *zerod* objects form an *oned* object. The wire-frame of one cut element in Figure B-1(b) is shown again in Figure B-3 with more patch-edge *oned*'s denoted. Then for each patch-edge *oned*, we need to determine its null state, i.e. whether it is outside the computational domain. We then discard the null *oned*'s; and for each non-null *oned*, we need to determine which background element (or face) it lies in, and this information will be used later for the construction of *twod* objects. For the example in Figure B-3, we need to mark $oned_5$ and $oned_8$ as null, and mark $oned_6$ and $oned_7$ as inside the only background element.

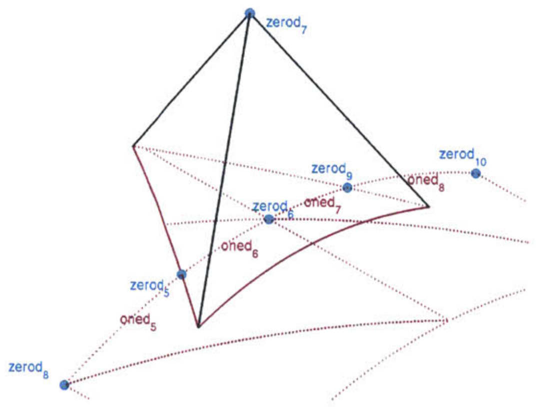


Figure B-3: Example of *oned* objects on patch edges

To deduce the null state for each patch-edge *oned*, we start from a patch-edge *oned* that intersects with a background face, for instance, $oned_5$ in Figure B-3, which intersects a background face at $zerod_5$. By evaluating the tangent vector of the patch edge at $zerod_5$, we can determine the null state of $oned_5$. Then we traverse through each patch edge, and deduce the null state of each other patch-edge *oned* based on topology. For example, traversing from $oned_5$ to $oned_6$ encounters an intersection point on a background face (with odd multiplicity), then these two *oned* objects have to be in the two neighbor background elements (or on the two sides of the domain boundary); traversing from *oned*$_6$ to *oned*$_7$ encounters no intersection point, and hence these two *oned*'s must be in the same background element. More logic is built into the algorithm if an encountered intersection point happens to be on a background edge or background vertex. There are still some cases, though rare, where tangent information is required. Figure B-4 shows such an example, where a patch-edge *oned* has both its endpoint *zerod* objects on background edges.

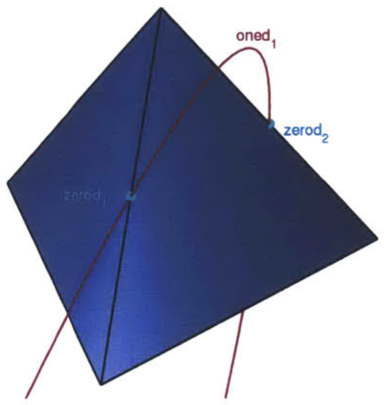


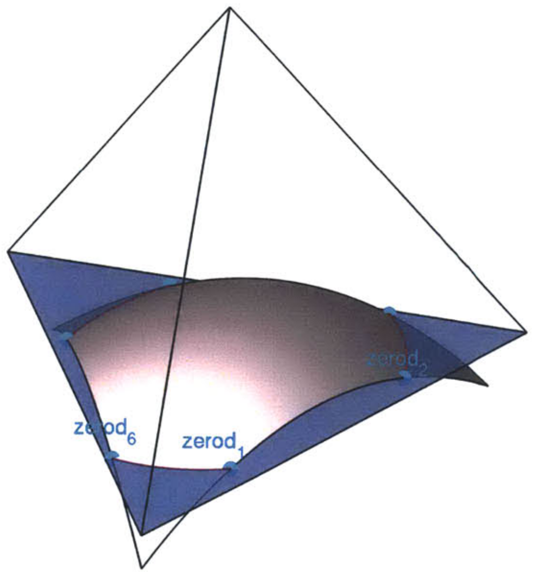
Figure B-4: Example where null state of patch-edge *oned* object cannot be deduced solely based on topology

Part 2: Background-Edge *oned* Objects

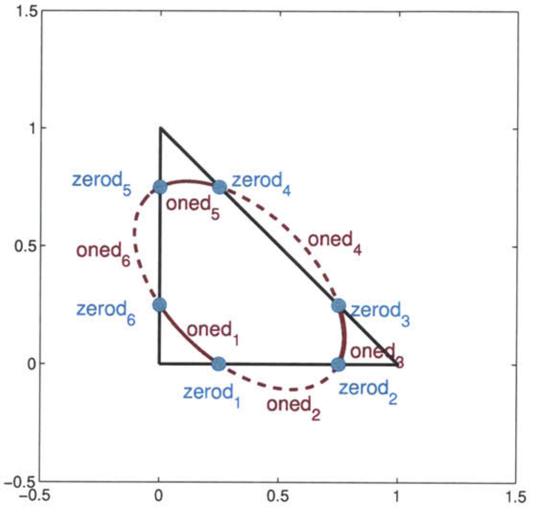
We then construct the *oned* objects on background edges, for example, *oned*$_3$ in Figure B-1(b). The *zerod* objects on each background edge are sorted and connected into *oned* objects. We also need to determine the null state of each formed background-edge *oned*, i.e. whether it is outside the computational domain. As described in Section 2.4, the null state of one *oned* is first determined by evaluating the patch normal vector, and the null state of each other *oned* can be deduced solely based on topology by traversing through each background edge. Again, there are still some cases, though rare, where the patch normal vector is required.

Part 3: Patch-Face *oned* Objects

We then construct the *oned* objects on patch faces, which are also named embedded *oned* objects. In patch reference space, they are defined by conic sections, as shown in Figure 2-3 in Chapter 2, where $oned_1$ and $oned_2$ are such examples. Thus, for the conic defined by each pair of a patch face and a background face, we collect all the associated *zerod*'s, and we need to connect them in order. An example is shown in Figure B-5, where we need to order six *zerod*'s on one conic.



(a) Physical space



(b) Patch reference space

Figure B-5: An example where the intersection curve between a patch and a background face has six *zerod* objects and three null embedded *oned* objects

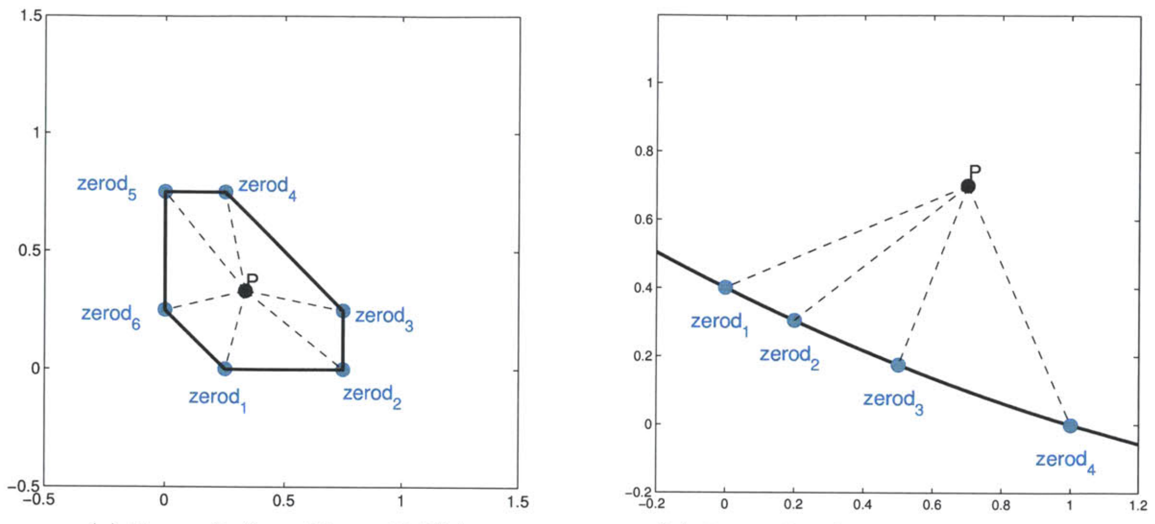
For conics with two branches (e.g. hyperbola), we first need to group the *zerod*'s into different branches. Achieving this efficiently using the adaptive precision arithmetic is described in Appendix A.2. We then sort the points along each branch of conic sections. As described in Section 2.3, this is done by exploiting the convexity of conic sections without the need for their parameterizations. On a convex segment, the order of points is the same as the order on the convex hull formed by these points [66]. Therefore, we can choose an origin to be the center of all the *zerod*'s that need to be ordered, and this origin is guaranteed to be inside the convex hull due to convexity of the conic section. One example is shown in Figure B-6(a), with the six *zerod*'s from Figure B-5(b). Then the ordering of the points can be deduced based on the angle of the vector pointing from the origin to each *zerod*. Note for conics that are not closed, this algorithm is still valid, but one constructed *oned* will in fact connect at an infinite distance. This does not cause an issue as this *oned* is certainly outside the unit triangle and will be discarded when the null state of each *oned* is inspected.

One special case is when the conic is degenerate or nearly-degenerate, shown in Figure B-6(b) for an example. A case is treated as nearly-degenerate if the angle between the vector from for example *zerod*$_1$ to every other *zerod* is smaller than 0.5°. Note this test is implemented in *double* precision. For such a case, choosing the origin to be the center of all *zerod*'s will lead to severe conditioning problems, and consequently efficiency issues when we sort the *zerod*'s using the adaptive precision arithmetic. Instead, we first find the two *zerod* objects that are the farthest from each other. Denote their coordinates by $\mathbf{v}_0$ and $\mathbf{v}_1$, then the origin is chosen as

$$\frac{1}{2}(\mathbf{v}_0 + \mathbf{v}_1) + \frac{1}{2}||\mathbf{v}_1 - \mathbf{v}_0||\hat{\mathbf{n}},$$

where $\hat{\mathbf{n}}$ is the unit vector such that $\hat{\mathbf{n}} \cdot (\mathbf{v}_1 - \mathbf{v}_0) = 0$.

After the *zerod* objects on the conic section are connected to *oned* objects, we then determine the null state of each *oned*, i.e. whether it is outside the patch face, or equivalently outside the unit triangle in the reference space. For the example in Figure B-5(b), we need to determine that three of the six *oned* objects are null. Same as for other *oned* types we have described, we first determine the null state of one *oned*, then the rest can be determined based on topology. For instance, the null state of *oned*$_1$ has to be different from *oned*$_2$ because of the intersection point *zerod*$_1$. In fact, null state of the first *oned* can also be deduced based on topology because a conic can intersect a triangle only in a limited number of ways. Therefore, no arithmetic is needed at all for this section except in some rare cases where a conic only intersects at patch vertices. Figure B-7 shows such an example, where we need to compute the



(a) Example from Figure B-5(b) (b) Example of nearly-degenerate conics

Figure B-6: Sorting points along conic sections

tangent vector at *zerod*$_2$ for deducing the null states of the two adjoining *oned* objects.

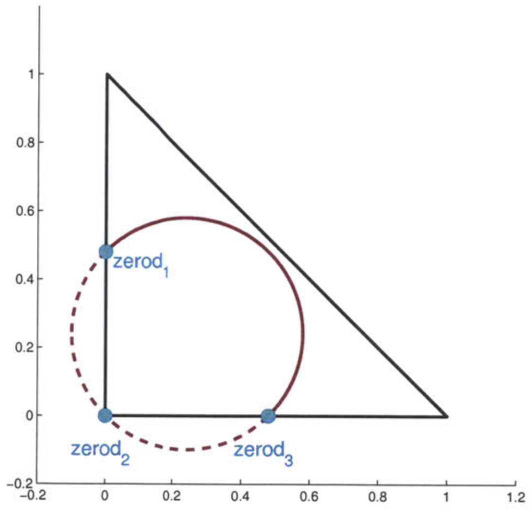
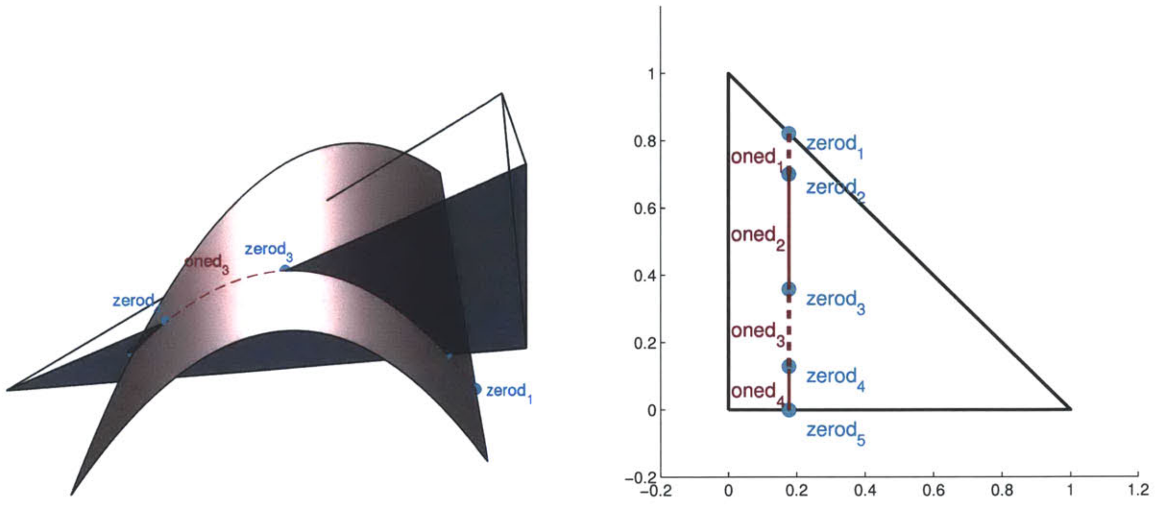


Figure B-7: A special case where determining the null state of embedded *oned* objects requires tangent computation

For certain cases, the embedded *oned* objects can be outside the background face, and such *oned* objects are also marked as null. An example is the *oned*$_3$ shown in Figure B-8. Again, we determine the null state of one *oned* based on tangent information, and march along the conic in the patch reference space to deduce the rest *oned*'s.



(a) Physical space (b) Patch reference space

Figure B-8: Example where embedded *oned* objects are outside background face

B.3 Construction of Intersection Faces and Volumes (*twod* and *threed* Objects)

After all the *oned* objects are constructed, they are then used to form the intersection faces, referred to as the *twod* objects. There are two types of *twod* objects: on patch face and on background face. For the patch-face type, we collect all the embedded *oned*'s in each background element, and connect them into one or multiple loops. For the background-face type, we collect all the *oned*'s on each background face, and connect them into loops.

The last step in the intersection algorithm is to construct the intersection volumes, namely cut elements. For each background element, we collect all the associated *twod* objects, and join them to form closed volumes. Note each background element can have multiple cut elements. A pseudocode for the construction of *twod* and *threed* objects is given in Algorithm B.3.

At the end, all the intersection point coordinates and conic coefficients are converted to *double* precision. They define the geometry of each cut element, and will also be used to compute the associated quadrature rule.

```
Input: Background grid, quadratic-patch geometry, zerod objects, oned objects
Output: twod objects, threed objects
/* Patch-face twod objects                                                  */
for each background element $K_b$ do
    Collect all embedded oned objects inside $K_b$ and on its faces;
    Connect these oned objects into loops, and store each loop as a twod object;
end
/* Background-face twod objects                                             */
for each background face $f_b$ do
    Collect all oned objects on $f_b$ and on its edges;
    Connect these oned objects into loops, and store each loop as a twod object;
end
/* threed objects                                                           */
for each background element $K_b$ do
    Collect all twod objects inside $K_b$ and on its faces;
    Connect these twod objects into enclosed volumes, and store each volume as
    a threed object;
end
```

Algorithm B.3: Construction of *twod* and *threed* objects

B.4 Parallelization

We parallelize the whole intersection algorithm, together with the quadrature-rule generation described in Chapter 3. Before partitioning, each background element is tested for potential intersection based on its bounding box, and a large weight is assigned if the test is positive. The background mesh is then partitioned using ParMetis [70], which computes a partitioning such that the number of connections is minimized and that each partition has approximately the same amount of weights. The quadratic-patch definition of the geometry is copied to each processor, which then solves the intersection problem on the partitioned background mesh. The cut topology is guaranteed to be consistent across partitions, because the adaptive precision arithmetic ensures the cut topology on each partition to be equivalent to the theoretical result.

The cut mesh on each partition is then assembled on the root processor, and a global cut mesh is constructed. This global mesh is used when we apply the merging technique, which combines two (or more) cut elements into one element as explained in Section 4.2. Also, if a partitioned background mesh has no intersection with the geometry (e.g. Ω_3 in Figure B-9), we cannot easily determine whether the whole partition is inside the computational domain without communicating with other partitions. For such a partition, this information is thus deduced after the global cut mesh is constructed.

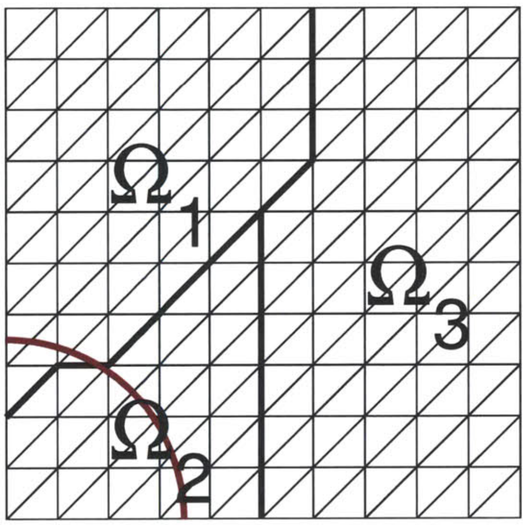


Figure B-9: An example partitioned background mesh, with one partition not intersecting the geometry

Appendix C

Notes on Cut-Cell Quadrature Rule

C.1 Quadrature Quality Measure

This section provides proofs for the theorems stated in Section 3.2.2. Again, for presentation brevity, we always assume the integration domain Ω has a unit volume. We first prove Theorem 3.4:

Proof. The proof for Item 3 of the theorem can be found in [63, 123], so we only prove Items 1 and 2 here. Based on Assumption 3.3, Eq. (3.4) gives

$$\mathbf{F}^{\infty} \equiv \lim_{n_q \to \infty} \mathbf{F} = \arg\min_{\mathbf{F}} \int_{\Omega} \left(\sum_{i=1}^{n_b} \mathbf{F}_i \psi_i(\mathbf{x}) - f(\mathbf{x}) \right)^2 d\mathbf{x},$$

which is the L^2 projection problem of $f(\mathbf{x})$ onto the space $\mathbb{P}_p^d$. The solution $\mathbf{F}^{\infty}$ thus satisfies

$$\int_{\Omega} f(\mathbf{x}) \psi_j(\mathbf{x}) d\mathbf{x} = \int_{\Omega} \sum_{i=1}^{n_b} \mathbf{F}_i^{\infty} \psi_i(\mathbf{x}) \psi_j(\mathbf{x}) d\mathbf{x}, \quad \forall \psi_j \in \mathbb{P}_p^d.$$

Since a constant function is in $\mathbb{P}_p^d$, we have

$$\int_{\Omega} f(\mathbf{x}) d\mathbf{x} = \int_{\Omega} \sum_{i=1}^{n_b} \mathbf{F}_i^{\infty} \psi_i(\mathbf{x}) d\mathbf{x} = \lim_{n_q \to \infty} \int_{\Omega} \sum_{i=1}^{n_b} \mathbf{F}_i \psi_i(\mathbf{x}) d\mathbf{x},$$

which proves Item 1.

From the definition of Q in (3.7), we have

$$Q = \sum_{q=1}^{n_q} \frac{1}{c_q} w_q^2 = 1 + \sum_{q=1}^{n_q} \frac{1}{c_q} (w_q - c_q)^2 > 1, \tag{C.1}$$

where the second equality uses the fact that

$$\sum_{q=1}^{n_q} w_q = 1, \quad \text{and} \quad \sum_{q=1}^{n_q} c_q = 1.$$

Note (C.1) shows that $Q - 1$ essentially measures the difference between $\{c_q\}$ and $\{w_q\}$. For the weights $\{w_q\}$ defined in (3.6), this difference makes $\{w_q\}$ a degree$-p$ quadrature rule regardless of the choice of $\{c_q\}$. Further, for $\{w_q\}$ in (3.6), the value of Q is given in (3.13):

$$Q = \mathbf{b}^T \left(\boldsymbol{V}\mathbf{C}\boldsymbol{V}^T\right)^{-1} \mathbf{b} \tag{C.2}$$

$$= \mathbf{b}^T \begin{bmatrix} \sum_q c_q \phi_1(\mathbf{x}_q)^2 & \cdots & \\ \vdots & \sum_q c_q \phi_i(\mathbf{x}_q)\phi_j(\mathbf{x}_q) & \\ & & \ddots \end{bmatrix}^{-1} \mathbf{b}. \tag{C.3}$$

Based on Assumption 3.3, Eq. (C.3) gives

$$\begin{aligned} \lim_{n_q \to \infty} Q &= \mathbf{b}^T \lim_{n_q \to \infty} \begin{bmatrix} \sum_q c_q \phi_1(\mathbf{x}_q)^2 & \cdots & \\ \vdots & \sum_q c_q \phi_i(\mathbf{x}_q)\phi_j(\mathbf{x}_q) & \\ & & \ddots \end{bmatrix}^{-1} \mathbf{b} \\ &= \mathbf{b}^T \mathbf{M}^{-1} \mathbf{b}, \end{aligned} \tag{C.4}$$

where $\mathbf{M}$ is the mass matrix, i.e. $\mathbf{M}_{ij} = \int_\Omega \psi_i(\mathbf{x})\psi_j(\mathbf{x}) d\mathbf{x}$. As (C.2) is independent of the basis polynomials, we can let $\{\psi_i\}$ be a set of orthonormal basis with $\psi_1(\mathbf{x}) = 1$.

Then $\mathbf{M}$ is the identity matrix, and $\mathbf{b} = [1, 0, 0, ...]^T$. Therefore, (C.4) gives

$$\lim_{n_q \to \infty} Q = 1.$$

□

The proof for Theorem 3.5 follows that for Theorem 4.1 in [119], and is given below.

Proof. Let $f_p^* \in \mathbb{P}_p^d$ denote the best approximation of f on $\overline{\Omega}$ with respect to $\|\cdot\|_\infty$. Let $I(\cdot)$ denote the integral $\int_\Omega (\cdot) d\mathbf{x}$, and let $I_p(\cdot)$ denote the approximation using the given degree-p quadrature rule $\{\mathbf{x}_q, w_q\}$. Then we have

$$\begin{aligned} |I(f) - I_p(f)| &= |I(f - f_p^*) - I_p(f - f_p^*)| \\ &\le |I(f - f_p^*)| + |I_p(f - f_p^*)| \\ &\le \|f - f_p^*\|_\infty + \sum_q |w_q| \|f - f_p^*\|_\infty \\ &= \|f - f_p^*\|_\infty (1 + \sum_q |w_q|) \\ &\le \|f - f_p^*\|_\infty (1 + \sqrt{Q}), \end{aligned}$$

where the last inequality is due to

$$\left(\sum_q |w_q| \right)^2 = \left(\sum_q \sqrt{c_q} \frac{1}{\sqrt{c_q}} |w_q| \right)^2 \le \sum_q c_q \cdot \sum_q \frac{1}{c_q} w_q^2 = Q.$$

□

C.2 Parameterization of Conics

This section reviews the parameterizations we use for integration of polynomials along conics. For a general conic, we use the parameterization with the property that a point sequence from evenly spaced parameters forms a polygon that encloses the maximum inscribed area [101]. More specifically, for an ellipse with a center $\mathbf{O}$, a semi-major axis of a along $\mathbf{X}$, and a semi-minor axis of b along $\mathbf{Y}$, the parameterization is

$$\mathbf{C}(u) = \mathbf{O} + a\cos u\mathbf{X} + b\sin u\mathbf{Y}; \tag{C.5}$$

similarly, a hyperbola is given by

$$\mathbf{C}(u) = \mathbf{O} - a\cosh u\mathbf{X} - b\sinh u\mathbf{Y}; \tag{C.6}$$

and for a parabola with a vertex $\mathbf{O}$, an axis $\mathbf{X}$ with a focal length a, a tangent direction $\mathbf{Y}$ at the vertex, the parameterization is

$$\mathbf{C}(u) = \mathbf{O} + au^2\mathbf{X} + 2au\mathbf{Y}. \tag{C.7}$$

This parameterization can be very sensitive to precision when the conic section is close to being a straight line segment. This is demonstrated in the following test case, where we compute the arc length of a circular arc centered at (u, u), $u \geq 1$, and passing by $(1, 0)$ and $(0, 1)$, as shown in Figure C-1. As u increases, the arc approaches to a straight line segment. The standard Legendre-Gauss integration rule of an order 39 is used in the parameter space. The dashed line in Figure C-2 shows the error using this parameterization, and as the arc becomes close to being a straight segment, the quadrature error rapidly increases.

In this work, a "nearly straight" conic section is detected by computing the angle between the normal vectors at the end points, $\mathbf{v}_0$ and $\mathbf{v}_1$. If the angle is less than 5°, we use the polar parameterization described below; more details can be found on page 87 of [88]. With respect to an origin $\mathbf{O}$, let a general conic be

$$Ax^2 + 2Bxy + Cy^2 + 2Dx + 2Ey + F = 0.$$

The polar representation can be found by substituting $x = r\cos\theta$ and $y = r\sin\theta$, and we then obtain

$$1/r = T_1 \sin\theta + T_2 \cos\theta \pm \sqrt{T_3 \sin 2\theta + T_4 \cos 2\theta + T_5},$$

where

$$T_1 = -\frac{E}{F}, \quad T_2 = -\frac{D}{F}, \quad T_3 = \frac{DE - BF}{F^2},$$
$$T_4 = \frac{D^2 - AF - E^2 + CF}{2F^2}, \quad T_5 = \frac{D^2 - AF + E^2 - CF}{2F^2}.$$

The key in this parameterization is the choice of the origin. As we only use this parameterization for sections that are nearly straight, the origin is chosen to be

$$\mathbf{O} \equiv \frac{1}{2}(\mathbf{v}_0 + \mathbf{v}_1) + \|\mathbf{v}_1 - \mathbf{v}_0\|\hat{\mathbf{n}},$$

where $\hat{\mathbf{n}}$ is a unit vector such that $\hat{\mathbf{n}} \cdot (\mathbf{v}_1 - \mathbf{v}_0) = 0$. The error for the previous test case is in Figure C-2, which shows that this parameterization does not suffer when the section is close to being straight. Note for a general conic that is not nearly degenerate, choosing a proper origin point is not easy, and we still use the parameterization defined in (C.5) through (C.7). More specifically, if the line from the origin to a point on the conic section is tangent to the section, the parameterization can lead to a set of very poorly spaced quadrature points.

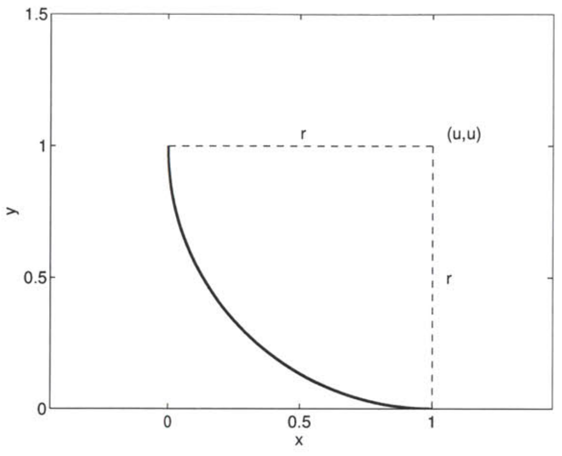


Figure C-1: Example for conic parameterization

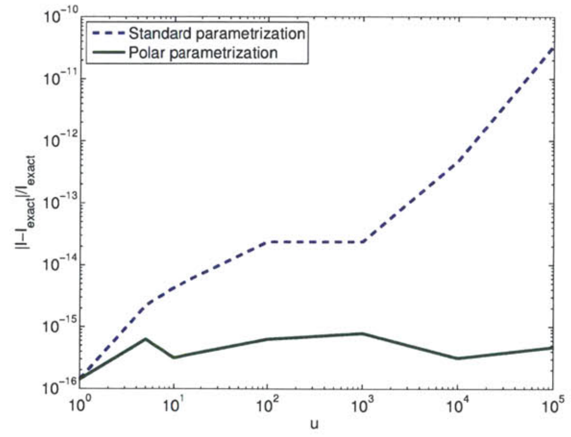


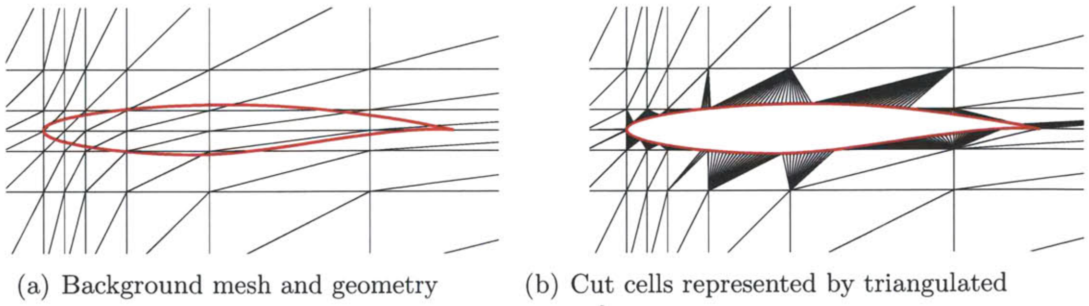
Figure C-2: Quadrature error for nearly-degenerate conic sections

Appendix D

Visualization of Cut Cells

D.1 Two Dimensions

To visualize cut-cell meshes in two dimensions, each cut element is represented by a polygon, whose vertices include all the *zerod* objects on the cut element. In addition, each *oned* object on the spline geometry is divided by ten additional points, which are evenly spaced in the parameter space of the spline. These additional points are also included in the polygon vertices. The polygon is then triangulated, and the result triangles are used for visualizing PDE solutions on the cut-cell mesh. One example of the triangulated polygons for cut-cell visualization is shown in Figure D-1.



(a) Background mesh and geometry (b) Cut cells represented by triangulated polygons

Figure D-1: Example for cut-cell visualization in two dimensions

D.2 Three Dimensions

For cut-cell meshes in three dimensions, we only developed visualization of results on the embedded geometry and on background boundary faces. More specifically, each quadratic patch is split into regions enclosed by conic lines in the reference space. An example can be found in Figure 2-3 in Chapter 2. For visualization, each of these regions is approximated by a polygon, whose vertices are defined by all the *zerod* objects on the region. The polygon is then triangulated in the reference space, and each triangle is used for visualization. For background boundary faces, each cut face is approximated by a polygon in the physical coordinate space, and the polygon is triangulated for visualization.

Appendix E

Governing Equations

This appendix describes the governing equations considered in this work: the advection-diffusion-reaction equation and the Navier-Stokes equations. They both have the form of a conservation law defined in Eq. (4.1), which is written again:

$$\frac{\partial u}{\partial t} + \nabla \cdot \mathcal{F}^i(u, \mathbf{x}, t) - \nabla \cdot \mathcal{F}^v(u, \nabla u, \mathbf{x}, t) = \mathcal{S}(u, \nabla u, \mathbf{x}, t), \quad \forall \mathbf{x} \in \Omega, \ t \in I,$$

where $u(\mathbf{x}, t) : \mathbb{R}^d \times \mathbb{R}^+ \rightarrow \mathbb{R}^m$ is the $m-$state solution vector.

E.1 Advection-Diffusion-Reaction Equation

For the advection-diffusion-reaction equation, the inviscid and viscous fluxes in the i-th coordinate direction, and the source term are defined as

$$\mathcal{F}^i_i = \beta_i u, \qquad \mathcal{F}^v_i = \kappa_{ij} \frac{\partial u}{\partial x_j}, \qquad \mathcal{S} = \gamma u,$$

where β_i is the advection velocity in the i-th coordinate direction, κ is the diffusivity tensor, and γ is the reaction coefficient. Note in this appendix, summation on repeated indices is implied unless otherwise stated.

E.2 Compressible Navier-Stokes Equations

E.2.1 Euler and Navier-Stokes Equation

For the compressible Navier-Stokes equations, the conservative state vector is $u = [\rho, \rho v_j, \rho E]^T$, where ρ is the density, v_j is the velocity in the j-th coordinate direction, and E is the specific total internal energy. The inviscid and viscous fluxes in the i-th coordinate direction are given by

$$\mathcal{F}_i^i \equiv \begin{bmatrix} \rho v_i \\ \rho v_j v_i + \delta_{ij} p \\ \rho H v_i \end{bmatrix}, \qquad \mathcal{F}_i^v \equiv \begin{bmatrix} 0 \\ \tau_{ij} \\ \tau_{ij} v_j + \kappa_T \frac{\partial T}{\partial x_i} \end{bmatrix}, \tag{E.1}$$

where p is the static pressure, $H = E + p/\rho$ is the specific total enthalpy, T is the temperature calculated from the ideal gas law, κ_T is the thermal conductivity, and τ is the shear stress, for which a Newtonian fluid is assumed:

$$\tau_{ij} = \mu \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right) + \delta_{ij} \lambda \frac{\partial v_k}{\partial x_k},$$

and μ is the dynamic viscosity, and $\lambda = -2/3\mu$ is the bulk viscosity coefficient. The pressure is related to the state vector by

$$p = (\gamma - 1)\rho \left(E - \frac{1}{2} v_i v_i \right),$$

where γ is the ratio of specific heats. The dynamic viscosity is modeled using Sutherland's law:

$$\mu = \mu_{\text{ref}} \left(\frac{T}{T_{\text{ref}}} \right)^{1.5} \frac{T_{\text{ref}} + T_s}{T + T_s}, \tag{E.2}$$

unless otherwise stated. The thermal conductivity, κ_T, is related to μ by the Prandtl number, Pr, according to $\kappa_T = c_p \frac{\mu}{Pr}$, where c_p is the specific heat at constant pressure.

E.2.2 Reynolds-Averaged Navier-Stokes Equations

The Reynolds-Averaged Navier-Stokes (RANS) Equations are derived by temporally averaging the Navier-Stokes equations with Favre averaging procedure. In this work, the closure of the RANS equation is accomplished by the Spalart-Allmaras (SA) turbulence model [115]. The specific form of the model is based on the work of Oliver [92], where two modifications to the original SA model were made. The first is a generalization of the model to compressible flows, and the second is a set of modifications intended to improve the robustness for higher-order simulations.

The conservative state vector for the RANS-SA equations corresponds to the mean flow states, and is denoted by $u = [\rho, \rho v_j, \rho E, \rho\tilde{\nu}]^T$, where $\tilde{\nu}$ is the working variable for the SA model, and is algebraically related to the eddy viscosity, μ_t:

$$\mu_t = \begin{cases} \rho\tilde{\nu} f_{v1}, & \tilde{\nu} \geq 0 \\ 0, & \tilde{\nu} < 0 \end{cases},$$

where

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}, \quad \chi = \frac{\tilde{\nu}}{\nu},$$

and $\nu = \mu/\rho$ is the kinematic viscosity. The inviscid and viscous fluxes of the RANS-SA system in the i-th coordinate direction are given by

$$\mathcal{F}_i^i = \begin{bmatrix} \rho v_i \\ \rho v_j v_i + \delta_{ij} p \\ \rho H v_i \\ \rho\tilde{\nu} \end{bmatrix}, \qquad \mathcal{F}_i^v = \begin{bmatrix} 0 \\ \tau_{ij}^{\mathrm{RANS}} \\ \tau_{ij}^{\mathrm{RANS}} v_j + \kappa_T^{\mathrm{RANS}} \frac{\partial T}{\partial x_i} \\ \frac{1}{\sigma}\eta\frac{\partial\tilde{\nu}}{\partial x_i} \end{bmatrix},$$

where the effective shear stress, τ^{RANS}, and the thermal conductivity, κ_T^{RANS} are

$$\tau^{\mathrm{RANS}} = (\mu + \mu_t)\left[\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) + \delta_{ij}\lambda\frac{\partial v_k}{\partial x_k}\right]$$
$$\kappa_T^{\mathrm{RANS}} = c_p\left(\frac{\mu}{Pr} + \frac{\mu_t}{Pr_t}\right),$$

and the diffusion coefficient for the SA equation, η, is

$$\eta = \begin{cases} \mu(1+\chi), & \chi \geq 0 \\ \mu(1+\chi+\frac{1}{2}\chi^2), & \chi < 0 \end{cases}.$$

The source term of the RANS-SA system is

$$\mathcal{S} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ P - D + c_{b2}\rho\frac{\partial\tilde{\nu}}{\partial x_k}\frac{\partial\tilde{\nu}}{\partial x_k} \end{bmatrix}.$$

Here, the production term, P, is

$$P = \begin{cases} c_{b1}\tilde{S}\rho\tilde{\nu}, & \chi \geq 0 \\ c_{b1}S\rho\tilde{\nu}g_n, & \chi < 0 \end{cases},$$

where $g_n = 1 - f_{g_n}\chi^2/(1+\chi^2)$, $f_{g_n} = 10^5$, and

$$\tilde{S} = \begin{cases} S + \bar{S}, & \bar{S} \geq -c_{v2}S \\ S + \frac{S(c_{v2}^2 S + c_{v3}\bar{S})}{(c_{v3}-2c_{v2})S - \bar{S}}, & \bar{S} < -c_{v2}S \end{cases},$$

and $S = \sqrt{2\Omega_{ij}\Omega_{ij}}$ is the magnitude of the vorticity, $\Omega_{ij} = \frac{1}{2}(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i})$, and the near-wall correction term is given by

$$\bar{S} = \frac{\tilde{\nu} f_{v2}}{\kappa^2 d^2} \quad \text{with} \quad f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}},$$

where d is the distance to the nearest wall. The destruction term, D, is given by

$$D = \begin{cases} c_{w1} f_w \frac{\rho \tilde{\nu}^2}{d^2}, & \chi \geq 0 \\ -c_{w1} \frac{\rho \tilde{\nu}^2}{d^2}, & \chi < 0 \end{cases},$$

where

$$f_w = g \left(\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6} \right)^{1/6}, \quad g = r + c_{w2}(r^6 - r), \quad \text{and} \quad r = \frac{\tilde{\nu}}{\tilde{S} \kappa^2 d^2}.$$

The constants of the turbulence model are set to: $c_{b1} = 0.1355$, $\sigma = 2/3$, $c_{b2} = 0.622$, $\kappa = 0.41$, $c_{w1} = c_{b1}/\kappa^2 + (1 + c_{b2})/\sigma$, $c_{w2} = 0.3$, $c_{w3} = 2$, $c_{v1} = 7.1$, $c_{v2} = 0.7$, $c_{v3} = 0.9$, and $Pr_t = 0.9$.

Appendix F

Discontinuous Galerkin Method and Solution Strategy

This appendix describes the discontinuous Galerkin (DG) discretization and the discrete solution strategy for the conservation law defined in Eq. (4.1), which is written again:

$$\frac{\partial u}{\partial t} + \nabla \cdot \mathcal{F}^i(u, \mathbf{x}, t) - \nabla \cdot \mathcal{F}^v(u, \nabla u, \mathbf{x}, t) = \mathcal{S}(u, \nabla u, \mathbf{x}, t), \quad \forall \mathbf{x} \in \Omega,\ t \in I,$$

with the boundary conditions

$$\mathcal{B}(u, \mathcal{F}^v(u, \nabla u, \mathbf{x}, t) \cdot \hat{\mathbf{n}}, x, t; \mathrm{BC}) = 0, \quad \forall \mathbf{x} \in \partial\Omega,\ t \in I,$$

where $u(\mathbf{x}, t) : \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}^m$ is the $m-$state solution vector, $\mathcal{F}^i$, $\mathcal{F}^v$, and $\mathcal{S}$ denote the inviscid flux, viscous flux, and the source term, respectively. Note in this appendix, the dependency of the fluxes and the source term on $\mathbf{x}$ and t are always implied, and will not be explicitly stated.

F.1 Discontinuous Galerkin Discretization

The weak form from the spatial DG discretization reads: find $u \in V_{h,p}$ such that

$$\mathcal{R}_{h,p}(u_{h,p}, v) = 0, \quad \forall v \in V_{h,p}, \tag{F.1}$$

where $\mathcal{R}_{h,p} : V_{h,p} \times V_{h,p} \to \mathbb{R}$ consists of discretizations of the inviscid term, viscous term, and the source term:

$$\mathcal{R}_{h,p}(u_{h,p}, v) = \mathcal{R}^i_{h,p}(u_{h,p}, v) + \mathcal{R}^v_{h,p}(u_{h,p}, v) + \mathcal{R}^s_{h,p}(u_{h,p}, v).$$

The DG discretization for the inviscid term is given by

$$\begin{aligned}\mathcal{R}^i_{h,p}(u_{h,p}, v) = &- \sum_{K \in \mathcal{T}_h} \int_K \nabla v^T \cdot \mathcal{F}^i(u_{h,p}) d\Omega + \sum_{e \in \Gamma_I} \int_e (v^+ - v^-)^T \mathcal{H}(u^+_{h,p}, u^-_{h,p}; \hat{\mathbf{n}}^+) ds \\ &+ \sum_{e \in \partial\Omega} \int_e v^{+T} \mathcal{H}^b(u^+_{h,p}, u_b(u^+_{h,p}; \mathrm{BC}); \hat{\mathbf{n}}^+) ds,\end{aligned}$$

where $(\cdot)^+$ and $(\cdot)^-$ denote the trace values on the two sides of a face e, $\hat{\mathbf{n}}^+$ is the normal vector pointing from $+$ to $-$, and $\mathcal{H}$ and $\mathcal{H}^b$ are numerical flux functions on interior and boundary faces, respectively. On boundary faces, the interior side is always the $+$ side by convention, and the boundary state, u_b, is in general a function of the interior state and the boundary conditions. In this work, the numerical flux $\mathcal{H}$ uses Roe's approximate Riemann solver [106]:

$$\mathcal{H}(u^+_{h,p}, u^-_{h,p}; \hat{\mathbf{n}}^+) = \frac{1}{2}\left(\hat{\mathbf{n}}^+ \cdot \mathcal{F}^i(u^+_{h,p}) + \hat{\mathbf{n}}^- \cdot \mathcal{F}^i(u^-_{h,p})\right) + \frac{1}{2}|\mathcal{A}^{\mathrm{Roe}}(u^+_{h,p}, u^-_{h,p}; \hat{\mathbf{n}}^+)|(u^+_{h,p} - u^-_{h,p}),$$

where $\mathcal{A}^{\mathrm{Roe}}$ is the flux Jacobian matrix computed at the Roe's mean state. The implementation for the boundary conditions follows the work [92].

The viscous term is discretized according to the second form of Bassi and Rebay (BR2) [17]. Let the viscous flux have the form of

$$\mathcal{F}^v(u, \nabla u) = \mathcal{A}(u)\nabla u,$$

and then the semi-linear form for the viscous term is given by

$$\begin{aligned}
\mathcal{R}^v_{h,p}(u_{h,p}, v) = &\sum_{K\in\mathcal{T}_h}\int_K \nabla v^T \cdot \mathcal{A}(u_{h,p})\nabla u_{h,p} d\Omega \\
&- \sum_{e\in\Gamma_I}\int_e \left[\{\mathcal{A}^T(u_{h,p})\nabla v\}^T \cdot [\![u_{h,p}]\!] + [\![v]\!]^T \cdot \{\mathcal{A}(u_{h,p})(\nabla u_{h,p} + \eta_e r^e([\![u_{h,p}]\!]))\}\right] ds \\
&- \sum_{e\in\partial\Omega}\int_e \Big[(\hat{\mathbf{n}}^+ \cdot \mathcal{A}_b^T \nabla v^+)^T(u^+_{h,p} - u_b) \\
&\qquad + v^{+^T}\hat{\mathbf{n}}^+ \cdot \mathcal{A}_b(\nabla u_b + \eta_e r^e((u^+_{h,p} - u_b)\hat{\mathbf{n}}^+))\Big] ds,
\end{aligned}$$

where $u_b = u_b(u^+_{h,p}; \text{BC})$, $\mathcal{A}_b = \mathcal{A}_b(u_b; \text{BC})$, and $\nabla u_b = \nabla u_b(\nabla u^+_{h,p}; BC)$ are chosen to specify the boundary flux according to the boundary condition. In addition, η_e is the BR2 stabilization parameter, $r^e : [V_{h,p}(e)]^d \to [V_{h,p}]^d$ is the local lifting operator defined by

$$\int_{\Omega^e_h} \tau^T \cdot r^e(\phi) d\Omega = -\int_e \{\tau\}^T \cdot \phi ds, \quad \forall\phi, \tau \in [V_{h,p}]^d,$$

and Ω^e_h is the union of elements sharing the face e. The jump and average operators for scalar function x and vector function $\mathbf{y}$ are defined as

$$\begin{aligned}
[\![x]\!] &\equiv x^-\mathbf{n}^- + x^+\mathbf{n}^+, & [\![\mathbf{y}]\!] &\equiv \mathbf{y}^- \cdot \mathbf{n}^- + \mathbf{y}^+ \cdot \mathbf{n}^+, \\
\{x\} &\equiv \frac{1}{2}(x^- + x^+), & \{\mathbf{y}\} &\equiv \frac{1}{2}(\mathbf{y}^- + \mathbf{y}^+).
\end{aligned}$$

The source term is discretized using a mixed form of Bassi *et al.* [14], which is asymptotically dual-consistent [92]. The semi-linear form is given by

$$\mathcal{R}^s_{h,p}(u_{h,p}, v) = \sum_{K\in\mathcal{T}_h}\int_K v^T \mathcal{S}(u_{h,p}, \nabla u_{h,p} + r^{\text{glob}}(u_{h,p}))d\Omega,$$

where the global lifting operator $r^{\text{glob}} : V_{h,p} \to [V_{h,p}]^d$ is defined by

$$\sum_{K\in\mathcal{T}_h} \int_K \tau^T \cdot r^{\text{glob}}(u_{h,p}) d\Omega = -\sum_{e\in\Gamma_I} \int_e \{\tau\}^T \cdot [\![u_{h,p}]\!] ds$$
$$- \sum_{e\in\partial\Omega} \int_e \tau^T \cdot \hat{\mathbf{n}}(u_{h,p} - u_b) ds, \quad \forall \tau \in [V_{h,p}]^d.$$

Note the global lifting operator is related to the local lifting operator by

$$r^{\text{glob}}(u_{h,p}) = \sum_{e\in\Gamma_I} r^e([\![u_{h,p}]\!]) + \sum_{e\in\partial\Omega} r^e(\hat{\mathbf{n}}(u_{h,p} - u_b)).$$

F.2 Shock Capturing

Shock capturing in this work uses the PDE-based artificial viscosity model developed by Barter and Darmofal [13], with modifications by Yano [124]. In particular, a shock indicator that measures the local solution regularity is used as a forcing term in an diffusive PDE, which smoothly propagates the effect of discontinuity and generates a smooth artificial viscosity field.

The jump-based shock indicator for element K is given by

$$S_K(w) = \log\left(\frac{1}{|\partial K|} \int_{\partial K} \left| \frac{g(w^+) - g(w^-)}{\frac{1}{2}(g(w^+) + g(w^-))} \right| ds\right),$$

where $g(w)$ is a scalar quantity defined from the PDE solution for detecting solution discontinuity. To prevent addition of artificial diffusion in smooth regions or addition of excessive viscosity, a filter originally developed by Persson and Peraire [97] is applied to yield

$$\bar{S}_K(S_K) = \begin{cases} 0, & S_K \le S_0(p) - \Delta S \\ \frac{S_{\max}}{2}\left(1 + \sin\left(\frac{\pi(S_K - S_0)}{2\Delta S}\right)\right), & S_0(p) - \Delta S < S_K \le S_0 + \Delta S \\ S_{\max}, & S_0(p) + \Delta S < S_K \end{cases},$$

with a polynomial-degree-dependent function $S_0(p)$ and parameters $\Delta S = 0.5$ and $S_{\max} = 1$.

The indicator $\bar{S}_K(S_K)$ then used as a source term in the artificial viscosity PDE:

$$\begin{aligned} \frac{\partial \nu_{\text{art}}}{\partial t} &= \frac{\partial}{\partial x_i}\left(\frac{\eta_{ij}}{\tau}\frac{\partial \nu_{\text{art}}}{\partial x_j}\right) + \frac{1}{\tau}\left[\frac{\bar{h}}{p}\lambda_{\max}(u)\bar{S}_K(u) - \nu_{\text{art}}\right] && \text{in } \Omega \\ \frac{\eta_{ij}}{\tau}\frac{\partial \nu_{\text{art}}}{\partial x_j}n_i &= \sqrt{C_1 C_2 \frac{p\lambda_{\max}}{h_{\min}}}(n_i n_j H_{ij})(\nu_{\text{art},\infty} - \nu_{\text{art}}) && \text{on } \partial\Omega. \end{aligned} \tag{F.2}$$

Here, $H(x) = \mathcal{M}^{-1/2}(x)$ is the generalized length scale based on the metric-tensor, $\eta_{ij} = C_2 H_{ik} H_{kj}$ is the diffusion coefficient, $\tau = h_{\min}/(C_1 p \lambda_{\max}(u))$ is the time scale based on the maximum wave speed, $\lambda_{\max}(u)$, $h_{\min} = \min_i \lambda_i(H)$ is the minimum (anisotropic) element size, and $\bar{h} = (\det(H))^{1/d}$ is the volume based element size. The two constants are set to $C_1 = 3$ and $C_2 = 5$. The resulting artificial viscosity field, ν_{art}, is again filtered to remove artificial viscosity in smooth regions and to cap the maximum viscosity.

F.3 Discrete Solution Strategy

To obtain the discrete form of the discretized equation (F.1), a basis for the function space, $V_{h,p}$, is chosen. Denote this basis by $\{\phi_i\}$ for $i = 1, ..., N$, and let $\mathbf{U} \in \mathbb{R}^N$ be the vector of expansion coefficients for the solution $u_{h,p} \in V_{h,p}$, that is, $u_{h,p} = \sum_{i=1}^N \mathbf{U}_i \phi_i(\mathbf{x})$. Then the discrete form of (F.1) can be written as a system of algebraic equations: find $\mathbf{U}$ such that

$$\mathbf{R}_s(\mathbf{U}) = 0, \tag{F.3}$$

where $\mathbf{R}_s(\mathbf{U})$ is the discrete residual vector such that $\mathbf{R}_s(\mathbf{U})_i = \mathcal{R}_{h,p}(u_{h,p}, \phi_i)$.

The system defined in Eq. (F.3) is solved using a pseudo-time continuation and backward Euler time integration. Given a discrete solution $\mathbf{U}^n$, the solution after one time step, $\mathbf{U}^{n+1}$, is solved from

$$\mathbf{R}_t(\mathbf{U}^{n+1}) \equiv \mathbf{M}^t(\mathbf{U}^{n+1} - \mathbf{U}^n) + \mathbf{R}_s(\mathbf{U}) = 0, \tag{F.4}$$

where $\mathbf{M}^t$ is the mass matrix weighted by local time step Δt_K. The time step Δt_K is based on a global CFL number: $\Delta t_K = \mathrm{CFL}\frac{h_k}{\lambda_K}$, where h_K is a measure of element size, and λ_K is the maximum characteristic speed in K. A single step of Newton's method is applied for Eq. (F.4) at each time step:

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \left(\mathbf{M}^t + \left.\frac{\partial \mathbf{R}_s}{\partial \mathbf{U}}\right|_{\mathbf{U}^n}\right)^{-1} \mathbf{R}_s(\mathbf{U}^n). \tag{F.5}$$

The solution process is marched forward in time until $||\mathbf{R}_s(\mathbf{U}^n)||_2$ is smaller than a user-specified tolerance. The CFL number is updated at each time step based on a physicality check, and a line search based on the unsteady residual, $\mathbf{R}_t$, is implemented to improve the robustness of the continuation procedure [87].

The linear system in Eq. (F.5) is solved using a restarted GMRES algorithm [108, 109]. To expedite the convergence of the GMRES algorithm, the linear system is preconditioned with an in-place block-ILU(0) factorization [43] with minimum discarded fill reordering scheme [98]. For parallel applications, an additive Schwarz preconditioner with overlap is used [42].

Appendix G

Proofs for Discretization of Interface Problems

G.1 Boundedness and Stability of DG Scheme for Interface Problems

This section proves boundedness and stability of the bilinear form B_h defined in Eq. (6.20), which can also be written as

$$B_h(u_h, v) = \int_\Omega \nabla u_h \cdot \kappa \nabla v d\Omega - \int_{\Gamma_A} \left([\![u_h]\!] \cdot \{\kappa \nabla v\} + \{\kappa \nabla u_h\} \cdot [\![v]\!] \right) ds$$
$$+ \sum_{e \in \Gamma_A} \int_\Omega \eta_e \kappa r^e([\![u_h]\!]) \cdot r^e([\![v]\!]) d\Omega, \qquad \text{(G.1)}$$

where the local lifting operator $r^e(\cdot)$ is defined in (6.19). The proof is very similar to the analysis in [7]. We define space $V(h) \equiv V_{h,p} + (H^2(\Omega^{(1)}) \oplus H^2(\Omega^{(2)}))$, and we define the following semi-norms and norms:

$$|v|^2_{1,h} = \sum_K |v|^2_{1,K}, \quad |v|^2_* = \sum_{e \in \Gamma \cup \Sigma} \|r^e([\![v]\!])\|^2_{0,\Omega},$$
$$|\!|\!|v|\!|\!| = |v|^2_{1,h} + \sum_K h_K^2 |v|^2_{2,K} + |v|^2_*.$$

Boundedness

We show the boundedness of B_h with respect to $|||\cdot|||$:

$$B_h(w,v) \leq C_b|||w||| \, |||v|||, \quad \forall w, v \in V(h). \tag{G.2}$$

We first bound $B_h(w,v)$ by

$$B_h(w,v) \leq \sup_\Omega \kappa \left(\int_\Omega \left|\nabla w \cdot \nabla v\right| d\Omega + \int_{\Gamma_A} \left| [\![w]\!] \cdot \{\nabla v\} + \{w\} \cdot [\![v]\!] \right| ds \right.$$
$$\left. + \sum_{e \in \Gamma_A} \int_\Omega \eta_e \left| r^e([\![w]\!]) \cdot r^e([\![v]\!]) \right| d\Omega \right),$$

and the term in the parentheses is bounded by $C|||w||| \, |||v|||$ as proven in [7]. This proves the boundedness (G.2).

Stability

In this section we prove the stability of the bilinear form:

$$B_h(v,v) \geq C_s |||v|||^2, \quad \forall v \in V_{h,p}. \tag{G.3}$$

From Eq. (G.1), we first write $B_h(v,v)$ as

$$B_h(v,v) = \int_\Omega \kappa \left(\left|\nabla v + r([\![v]\!])\right|^2 - \left|r([\![v]\!])\right|^2 \right) d\Omega + \sum_{e \in \Gamma_A} \int_\Omega \eta_e \kappa \left| r^e([\![v]\!]) \right|^2 d\Omega, \tag{G.4}$$

where the lifting operator $r(\cdot)$ is defined in (6.12). Then we apply the arithmetic-geometric mean inequality, $2ab \leq a^2\epsilon + b^2/\epsilon$, on the first term of (G.4), we can bound (G.4) for any $\epsilon > 0$ by

$$B_h(v,v) \geq \inf_\Omega \kappa \cdot \left(|v|_{1,h}^2 (1-\epsilon) + (1 - 1/\epsilon) \|r([\![v]\!])\|_{0,\Omega}^2 \right) - \sup_\Omega \kappa \cdot \|r([\![v]\!])\|_{0,\Omega}^2$$
$$+ \inf_e \eta_e \inf_\Omega \kappa \cdot |v|_*^2. \tag{G.5}$$

On triangular meshes, we have from [7]:

$$\|r([\![v]\!])\|_{0,\Omega}^2 \le 3|v|_*^2, \quad \forall v \in V(h).$$

Then for any $\epsilon < 1$, (G.5) can be further bounded by

$$B_h(v,v) \ge \inf_\Omega \kappa \cdot (1-\epsilon)|v|_{1,h}^2 + \left(3\inf_\Omega \kappa \cdot (1 - 1/\epsilon) - 3\sup_\Omega \kappa + \inf_e \eta_e \inf_\Omega \kappa\right)|v|_*^2.$$

Since we can choose ϵ as close to 1 as possible, a sufficient condition for stability (G.3) is

$$\inf_e \eta_e > 3\frac{\sup_\Omega \kappa}{\inf_\Omega \kappa}.$$

Note this bound is very loose. A much tighter bound can be obtained by bounding each elemental contribution in (G.4), which is found to be

$$\eta_e > 3 \max_{K \in \Omega_h^e} \frac{\sup_K \kappa}{\inf_K \kappa}.$$

G.2 Adjoint Formulation for Interface Problems

In this section, we derive the dual problem of Eq. (6.1) and (6.2) with respect to the output defined in Eq. (6.22). We first form the Lagrangian:

$$\begin{aligned}\mathcal{L}(u,\psi) \equiv &\sum_i \int_{\Omega^{(i)}} g_\Omega^{(i)} u^{(i)} d\Omega - \int_{\partial\Omega} g_{\partial\Omega}\kappa\nabla u \cdot \hat{\mathbf{n}} ds - \int_\Sigma g_\Sigma \kappa^{(1)} \nabla u^{(1)} \cdot \hat{\mathbf{n}}^{(1)} ds \\ &- \sum_i \int_{\Omega^{(i)}} \psi^{(i)} \left(-\nabla \cdot \left(\kappa^{(i)} \nabla u^{(i)}\right) - f^{(i)}\right) d\Omega, \end{aligned} \tag{G.6}$$

where u satisfies Eq. (6.1) and (6.2). The dual problem is defined by

$$\mathcal{L}'[u](w,\psi) = 0 \quad \forall \text{ permissible } w,$$

i.e. $w \in W$, and $[\![w]\!] = 0$ and $[\![\kappa \nabla w]\!] = 0$ on Σ, and $w = 0$ on $\partial\Omega$. From Eq. (G.6), we have

$$
\begin{aligned}
\mathcal{L}'[u](w,\psi) =& \sum_i \int_{\Omega^{(i)}} g_\Omega^{(i)} w^{(i)} d\Omega - \int_{\partial\Omega} g_{\partial\Omega} \kappa \nabla w \cdot \hat{\mathbf{n}} ds - \int_\Sigma g_\Sigma \kappa^{(1)} \nabla w^{(1)} \cdot \hat{\mathbf{n}}^{(1)} ds \\
&+ \sum_i \int_{\Omega^{(i)}} \psi^{(i)} \nabla \cdot \left(\kappa^{(i)} \nabla w^{(i)} \right) d\Omega \\
=& \sum_i \int_{\Omega^{(i)}} g_\Omega^{(i)} w^{(i)} d\Omega - \int_{\partial\Omega} g_{\partial\Omega} \kappa \nabla w \cdot \hat{\mathbf{n}} ds - \int_\Sigma g_\Sigma \kappa^{(1)} \nabla w^{(1)} \cdot \hat{\mathbf{n}}^{(1)} ds \\
&+ \sum_i \left(\int_{\partial\Omega^{(i)}} \psi^{(i)} \kappa^{(i)} \nabla w^{(i)} \cdot \hat{\mathbf{n}}^{(i)} - w^{(i)} \kappa^{(i)} \nabla \psi^{(i)} \cdot \hat{\mathbf{n}}^{(i)} ds \right. \\
&+ \left. \int_{\Omega^{(i)}} w^{(i)} \nabla \cdot \left(\kappa^{(i)} \nabla \psi^{(i)} \right) d\Omega \right) \\
=& \sum_i \int_{\Omega^{(i)}} w^{(i)} \left(\nabla \cdot \left(\kappa^{(i)} \nabla \psi^{(i)} \right) + g_\Omega^{(i)} \right) d\Omega \\
&+ \sum_i \int_{\partial\Omega^{(i)} \setminus \Sigma} \left(\psi^{(i)} - g_{\partial\Omega} \right) \kappa^{(i)} \nabla w^{(i)} \cdot \hat{\mathbf{n}}^{(i)} ds \\
&+ \int_\Sigma \kappa \nabla w ([\![\psi]\!] - g_\Sigma \hat{\mathbf{n}}^{(1)}) ds - \int_\Sigma w [\![\kappa \nabla \psi]\!] ds = 0 \quad \forall \text{ permissible } w.
\end{aligned}
$$

Therefore, the continuous adjoint solution ψ satisfies

$$
\begin{aligned}
-\nabla \cdot (\kappa \nabla \psi) &= g_\Omega^{(i)} \text{ in } \Omega^{(i)} \quad \forall i \\
\psi &= g_{\partial\Omega} \text{ on } \partial\Omega \\
[\![\psi]\!] = g_\Sigma \hat{\mathbf{n}}^{(1)}, &\quad [\![\kappa \nabla \psi]\!] = 0 \text{ on } \Sigma.
\end{aligned}
$$

Bibliography

[1] Slimane Adjerid and Tao Lin. A p-th degree immersed finite element for boundary value problems with discontinuous coefficients. *Appl. Numer. Math.*, 59(6):1303 - 1321, 2009.

[2] M. J. Aftosmis, M. J. Berger, and J. M. Melton. Adaptive Cartesian mesh generation. In J. F. Thompson, B. K. Soni, and N. P. Weatherill, editors, *Handbook of Grid Generation.* CRC Press, 1998.

[3] Mark Ainsworth and Bill Senior. An adaptive refinement strategy for *hp*-finite element computations. *Appl. Numer. Math.*, 26:165–178, 1998.

[4] S. R. Allmaras. *A coupled Euler/Navier-Stokes algorithm for 2-D unsteady transonic shock/boundary-layer interaction.* PhD thesis, Massachusetts Institute of Technology, 1989.

[5] S. R. Allmaras and M. B. Giles. A second-order flux split scheme for the unsteady 2-D Euler equations on arbitrary meshes. AIAA 1987-1119-CP, 1987.

[6] D. N. Arnold. An interior penalty finite element method with discontinuous elements. *SIAM J. Numer. Anal.*, 19:742–760, 1982.

[7] Douglas N. Arnold, Franco Brezzi, Bernardo Cockburn, and L. Donatella Marini. Unified analysis of discontinuous Galerkin methods for elliptical problems. *SIAM J. Numer. Anal.*, 39(5):1749–1779, 2002.

[8] Ivo Babuška. The finite element method for elliptic equations with discontinuous coefficients. *Computing*, 5:207–213, 1970.

[9] Garth A. Baker. Finite element methods for elliptic equations using nonconforming elements. *Math. Comp.*, 31:45–59, 1977.

[10] Wolfgang Bangerth and Rolf Rannacher. *Adaptive Finite Element Methods for Differential Equations.* Birkhäuser Verlag, Basel, 2003.

[11] Maxime Barrault, Yvon Maday, Ngoc Cuong Nguyen, and Anthony T. Patera. An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. *C. R. Math. Acad. Sci. Paris*, 339(9):667 - 672, 2004.

[12] John W. Barrett and Charles M. Elliott. Fitted and unfitted finite-element methods for elliptic equations with smooth interfaces. *IMA J. Numer. Anal.*, 7(3):283–300, 1987.

[13] Garrett E. Barter and David L. Darmofal. Shock capturing with PDE-based artificial viscosity for DGFEM: Part I, formulation. *J. Comput. Phys.*, 229(5):1810–1827, 2010.

[14] F. Bassi, A. Crivellini, S. Rebay, and M. Savini. Discontinuous Galerkin solution of the Reynolds averaged Navier-Stokes and k-ω turbulence model equations. *Comput. & Fluids*, 34:507–540, May-June 2005.

[15] F. Bassi and S. Rebay. High-order accurate discontinuous finite element solution of the 2D Euler equations. *J. Comput. Phys.*, 138(2):251–285, 1997.

[16] F. Bassi and S. Rebay. A high-order discontinuous finite element method for the numerical solution of the compressible Navier-Stokes equations. *J. Comput. Phys.*, 131:267–279, 1997.

[17] F. Bassi and S. Rebay. GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations. In Karniadakis Cockburn and Shu, editors, *Discontinuous Galerkin Methods: Theory, Computation and Applications*, pages 197–208. Springer, Berlin, 2000.

[18] R. Becker and R. Rannacher. A feed-back approach to error control in finite element methods: Basic analysis and examples. *East-West J. Numer. Math.*, 4:237–264, 1996.

[19] R. Becker and R. Rannacher. An optimal control approach to a posteriori error estimation in finite element methods. In A. Iserles, editor, *Acta Numerica*. Cambridge University Press, 2001.

[20] R. P. Beyer and R. J. Leveque. Analysis of a one-dimensional model for the immersed boundary method. *SIAM J. Numer. Anal.*, 29(2):332–364, 1992.

[21] Houman Borouchaki, Paul Louis George, Frédéric Hecht, Patrick Laug, and Eric Saltel. Delaunay mesh generation governed by metric specifications. Part I algorithms. *Finite Elem. Anal. Des.*, 25(1-2):61–83, 1997.

[22] L. Bos, J.-P. Calvi, N. Levenberg, A. Sommariva, and M. Vianello. Geometric weakly admissible meshes, discrete least squares approximations and approximate Fekete points. *Math. Comp.*, 80(275):1623 – 1638, 2011.

[23] Len Bos, Stefano De Marchi, Alvise Sommariva, and Marco Vianello. Computing multivariate Fekete and Leja points by numerical linear algebra. *SIAM J. Numer. Anal.*, 48(5):1984–1999, 2010.

[24] Frank J. Bossen and Paul S. Heckbert. A pliant method for anisotropic mesh generation. In *5th Intl. Meshing Roundtable*, pages 63–74, Oct. 1996.

[25] Christoph Burnikel, Stefan Funke, Kurt Mehlhorn, Stefan Schirra, and Susanne Schmitt. A separation bound for real algebraic expressions. In *Lecture Notes in Computer Science*, pages 254–265. Springer, 2001.

[26] Christoph Burnikel, Kurt Mehlhorn, and Stefan Schirra. The LEDA class real number. Technical report, Max-Planck Institut Inform, 1996.

[27] Brian Camp, Tao Lin, Yanping Lin, and Weiwei Sun. Quadratic immersed finite element spaces and their approximation capabilities. *Adv. Comput. Math.*, 24:81–112, 2006.

[28] G. Chavent and G. Salzano. A finite element method for the 1D water flooding problem with gravity. *J. Comput. Phys.*, 42:307–344, 1982.

[29] A. T. Chen and J. R. Rice. On grid refinement at point singularities for h-p methods. *Internat. J. Numer. Methods Engrg.*, 33(1):39–57, 1992.

[30] B. Cockburn, S. Hou, and C. W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws IV: The multidimensional case. *Math. Comp.*, 54:545–581, 1990.

[31] B. Cockburn, G. Karniadakis, and C. Shu. The development of discontinuous Galerkin methods. In *Lecture Notes in Computational Science and Engineering*, volume 11. Springer, 2000.

[32] B. Cockburn, S. Y. Lin, and C. W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: One dimensional systems. *J. Comput. Phys.*, 84:90–113, 1989.

[33] B. Cockburn and C. W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for scalar conservation laws II: General framework. *Math. Comp.*, 52:411–435, 1989.

[34] B. Cockburn and C. W. Shu. The local discontinuous Galerkin method for time-dependent convection-diffusion systems. *SIAM J. Numer. Anal.*, 35(6):2440–2463, December 1998.

[35] B. Cockburn and C. W. Shu. The Runge-Kutta discontinuous Galerkin finite element method for conservation laws V: Multidimensional systems. *J. Comput. Phys.*, 141:199–224, 1998.

[36] R. Cools, I.P. Mysovskikh, and H.J. Schmid. Cubature formulae and orthogonal polynomials. *J. Comput. Appl. Math.*, 127(12):121 – 152, 2001.

[37] Ronald Cools. Constructing cubature formulae: the science behind the art. In *Acta Numerica*. Cambridge University Press, 1997.

[38] Ronald Cools. Advances in multidimensional integration. *J. Comput. Appl. Math.*, 149(1):1 – 12, 2002.

[39] Philip J. Davis. A construction of nonnegative approximate quadratures. *Math. Comp.*, 21(100):578–582, 1967.

[40] W. N. Dawes, P. C. Dhanasekaran, A. A. J. Demargne, W. P. Kellar, and A. M. Savill. Reducing bottlenecks in the CAD-to-mesh-to-solution cycle time to allow CFD to participate in design. *Journal of Turbomachinery*, 123(11):552–557, 2001.

[41] L. Demkowicz, Ph. Devloo, and J. T. Oden. On an h-type mesh-refinement strategy based on minimization of interpolation errors. *Comput. Methods Appl. Mech. Engrg.*, 53:67–89, 1985.

[42] Laslo T. Diosady. *Domain Decomposition Preconditioners for Higher-Order Discontinuous Galerkin Discretizations.* PhD thesis, Massachusetts Institute of Technology, September 2011.

[43] Laslo T. Diosady and David L. Darmofal. Preconditioning methods for discontinuous Galerkin solutions of the Navier-Stokes equations. *J. Comput. Phys.*, 228:3917–3935, 2009.

[44] Thomas Dubé and Chee-Keng Yap. A basis for implementing exact geometric algorithms (extended abstract), 1993.

[45] David Eberly. Intersection of convex objects: the method of separating axis, 2001.

[46] Ioannis Z. Emiris and Elias P. Tsigaridas. Real algebraic numbers and polynomial systems of small degree. *Theoret. Comput. Sci.*, 409:186–199, December 2008.

[47] Ronald P Fedkiw, Tariq Aslam, Barry Merriman, and Stanley Osher. A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *J. Comput. Phys.*, 152(2):457 – 492, 1999.

[48] Krzysztof J. Fidkowski. A high-order discontinuous Galerkin multigrid solver for aerodynamic applications. Master's thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2004.

[49] Krzysztof J. Fidkowski. *A Simplex Cut-Cell Adaptive Method for High-Order Discretizations of the Compressible Navier-Stokes Equations.* PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2007.

[50] Krzysztof J. Fidkowski and David L. Darmofal. A triangular cut-cell adaptive method for higher-order discretizations of the compressible Navier-Stokes equations. *J. Comput. Phys.*, 225:1653–1672, 2007.

[51] Krzysztof J. Fidkowski, Todd A. Oliver, James Lu, and David L. Darmofal. p-Multigrid solution of high-order discontiunous Galerkin discretizations of the compressible Navier-Stokes equations. *J. Comput. Phys.*, 207(1):92–113, 2005.

[52] Frédéric Gibou and Ronald Fedkiw. A fourth order accurate discretization for the Laplace and heat equations on arbitrary domains, with applications to the Stefan problem. *J. Comput. Phys.*, 202(2):577 – 601, 2005.

[53] M. B. Giles and E. Süli. Adjoint methods for PDEs: a posteriori error analysis and postprocessing by duality. In *Acta Numerica*, volume 11, pages 145–236, 2002.

[54] Roland Glowinski, Tsorng-Whay Pan, and Jacques Periaux. A fictitious domain method for Dirichlet problem and applications. *Comput. Methods Appl. Mech. Engrg.*, 111(34):283 – 303, 1994.

[55] Leonidas J. Guibas. Implementing geometric algorithms robustly. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 15–22. Springer Berlin Heidelberg, 1996.

[56] Grégory Guyomarc'h, Chang-Ock Lee, and Kiwan Jeon. A discontinuous Galerkin method for elliptic interface problems with application to electroporation. *Comm. Numer. Methods Engrg.*, 25(10):991–1008, 2009.

[57] Wagdi G. Habashi, Julien Dompierre, Yves Bourgault, Djaffar Ait-Ali-Yahia, Michel Fortin, and Marie-Gabrielle Vallet. Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent CFD. part I: general principles. *Internat. J. Numer. Methods Fluids*, 32:725–744, 2000.

[58] Anita Hansbo and Peter Hansbo. An unfitted finite element method, based on Nitsche's method, for elliptic interface problems. *Comput. Methods Appl. Mech. Engrg.*, 191(4748):5537 – 5552, 2002.

[59] F. Hecht. Bamg: Bidimensional anisotropic mesh generator, 1998. `http://www-rocq1.inria.fr/gamma/cdrom/www/bamg/eng.htm`.

[60] Don Herbison-Evans. Solving quartics and cubics for graphics. In Alan W. Paeth, editor, *Graphics Gems V*, pages 3–15. Academic Press, San Diego, CA, USA, 1995.

[61] Christoph M. Hoffmann. Robustness in geometric computations. Technical report, Purdue University, 2001.

[62] Jianguo Huang and Jun Zou. A mortar element method for elliptic problems with discontinuous coefficients. *IMA J. Numer. Anal.*, 22(4):549–576, 2002.

[63] Daan Huybrechs. Stable high-order quadrature rules with equidistant points. *J. Comput. Appl. Math.*, 231(2):933–947, 2009.

[64] Hua Ji, Fue-Sang Lien, and Eugene Yee. An efficient second-order accurate cut-cell method for solving the variable coefficient Poisson equation with jump conditions on irregular domains. *Internat. J. Numer. Methods Fluids*, 52(7):723–748, 2006.

[65] C. Johnson and J. Pitkäranta. An analysis of the discontinuous Galerkin method for a scalar hyperbolic equation. *Math. Comp.*, 46:1–26, 1986.

[66] J. Johnstone. *The sorting of points along an algebraic curve.* PhD thesis, Cornell University, 1987.

[67] R. Kafafy, T. Lin, Y. Lin, and J. Wang. Three-dimensional immersed finite element methods for electric field simulation in composite materials. *Internat. J. Numer. Methods Engrg.*, 64(7):940–972, 2005.

[68] Michael Karasick, Derek Lieber, and Lee R. Nackman. Efficient Delaunay triangulation using rational arithmetic. *ACM Trans. Graph.*, 10:71–91, January 1991.

[69] Steve L. Karman. SPLITFLOW: A 3D unstructured Cartesian/prismatic grid CFD code for complex geometries. AIAA 1995-0343, 1995.

[70] George Karypis. Parmetis: Parallel graph partitioning and sparse matrix ordering library, 2006. `http://glaros.dtc.umn.edu/gkhome/views/metis/parmetis`.

[71] Mats G. Larson, Robert Sderlund, and Fredrik Bengzon. Adaptive finite element approximation of coupled flow and transport problems with applications in heat transfer. *Internat. J. Numer. Methods Fluids*, 57(9):1397–1420, 2008.

[72] Tobias Leicht and Ralf Hartmann. Anisotropic mesh refinement for discontinuous Galerkin methods in two-dimensional aerodynamic flow simulations. *Internat. J. Numer. Methods Fluids*, 56:2111–2138, 2008.

[73] Randall J. LeVeque and Zhilin Li. The immersed interface method for elliptic equations with discontinuous coefficients and singular sources. *SIAM J. Numer. Anal.*, 31(4):1019–1044, 1994.

[74] Z. Li, T. Lin, Y. Lin, and R. C. Rogers. An immersed finite element space and its approximation capability. *Numer. Methods Partial Differential Equations*, 20(3):338–367, 2004.

[75] Zhilin Li. The immersed interface method using a finite element formulation. *Appl. Numer. Math.*, 27(3):253 – 267, 1998.

[76] Zhilin Li and Kazufumi Ito. *The Immersed Interface Method: Numerical Solutions of PDEs Involving Interfaces and Irregular Domains (Frontiers in Applied Mathematics)*. Society for Industrial and Applied Mathematic, Philadelphia, 2006.

[77] Mark N. Linnick and Hermann F. Fasel. A high-order immersed interface method for simulating unsteady incompressible flows on irregular domains. *J. Comput. Phys.*, 204(1):157 – 192, 2005.

[78] Eric Hung-Lin Liu. Optimization and validation of discontinuous Galerkin code for the 3D Navier-Stokes equations. Master's thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, January 2011.

[79] Adrien Loseille and Frédéric Alauzet. Optimal 3D highly anisotropic mesh adaptation based on the continuous mesh framework. In *Proceedings of the 18th International Meshing Roundtable*, pages 575–594. Springer Berlin Heidelberg, 2009.

[80] Adrien Loseille and Frédéric Alauzet. Continuous mesh framework part I: Well-posed continuous interpolation error. *SIAM J. Numer. Anal.*, 49(1):38–60, 2011.

[81] Adrien Loseille and Frédéric Alauzet. Continuous mesh framework part II: Validations and applications. *SIAM J. Numer. Anal.*, 49(1):61–86, 2011.

[82] James Lu. *An a Posteriori Error Control Framework for Adaptive Precision Optimization Using Discontinuous Galerkin Finite Element Method.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2005.

[83] Yvon Maday, Ngoc Cuong Nguyen, Anthony T Patera, and George SH Pau. A general, multipurpose interpolation procedure: the magic points. *Comm. Pure Appl. Math.*, 8(1):383–404, 2009.

[84] D. J. Mavriplis. Results from the 3rd Drag Prediction Workshop using the NSU3D unstructured mesh solver. AIAA 2007-256, 2007.

[85] Todd Michal and Joshua Krakos. Anisotropic mesh adaptation through edge primitive operations. AIAA 2012-159, 2012.

[86] Victor Milenkovic. Robust polygon modeling. *Computer-Aided Design*, 25:546–566, 1993.

[87] James M. Modisette. *An Automated Reliable Method for Two-Dimensional Reynolds-averaged Navier-Stokes Simulations.* PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, September 2011.

[88] Forest Ray Moulton. *An introduction to celestial mechanics.* Dover Publications, 1984.

[89] S. E. Mousavi, H. Xiao, and N. Sukumar. Generalized Gaussian quadrature rules on arbitrary polygons. *Internat. J. Numer. Methods Engrg.*, 82(1):99–113, 2010.

[90] Scott M. Murman, Michael J. Aftosmis, and Stuart E. Rogers. Characterization of space shuttle ascent debris aerodynamics using CFD methods. AIAA 2005-1223, 2005.

[91] Sundararajan Natarajan, Stéphane Bordas, and D. Roy Mahapatra. Numerical integration over arbitrary polygonal domains based on Schwarz-Christoffel conformal mapping. *Internat. J. Numer. Methods Engrg.*, 80:103–134, 2009.

[92] Todd A. Oliver. *A Higher-Order, Adaptive, Discontinuous Galerkin Finite Element Method for the Reynolds-averaged Navier-Stokes Equations.* PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2008.

[93] C. Péniguel. Heat transfer simulation for industrial applications: needs, limitations, expectations. *Internat. J. Heat Fluid Flow*, 19(2):102 – 114, 1998.

[94] Xavier Pennec, Pierre Fillard, and Nicholas Ayache. A Riemannian framework for tensor computing. *Int. J. Comput. Vision*, 66(1):41–66, 2006.

[95] J. Peraire, M. Vahdati, K. Morgan, and O. C. Zienkiewicz. Adaptive remeshing for compressible flow computations. *J. Comput. Phys.*, 72:449–466, 1987.

[96] Jaime Peraire and Per-Olof Persson. The compact discontinuous Galerkin (CDG) method for elliptic problems. *SIAM J. Sci. Comput.*, 30(4):1806–1824, 2008.

[97] Per-Olof Persson and Jaime Peraire. Sub-cell shock capturing for discontinuous Galerkin methods. AIAA 2006-0112, 2006.

[98] Per-Olof Persson and Jaime Peraire. Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier-Stokes equations. *SIAM J. Sci. Comput.*, 30(6):2709–2722, 2008.

[99] Charles S. Peskin. The immersed boundary method. In *Acta Numerica*, pages 479–517, 2002.

[100] Jrg Peters and Ulrich Reif. The 42 equivalence classes of quadratic surfaces in affine n-space. *Comput. Aided Geom. Design.*, 15(5):459 – 473, 1998.

[101] Les A. Pieql and Wayne Tiller. *The NURBS Book (Monographs in Visual Communication).* Springer, 2 edition, 1996.

[102] James W. Purvis and John E. Burkhalter. Prediction of critical Mach number for store configurations. *AIAA Journal*, 17(11):1170–1177, 1979.

[103] W. H. Reed and T. R. Hill. Triangular mesh methods for the neutron transport equation. Technical Report LA-UR-73-479, Los Alamos Scientific Laboratory, 1973.

[104] G. R. Richter. An optimal-order error estimate for the discontinuous Galerkin method. *Math. Comp.*, 50:75–88, 1988.

[105] Th. Richter. Goal-oriented error estimation for fluid-structure interaction problems. *Comput. Methods Appl. Mech. Engrg.*, 223-224:28 – 42, 2012.

[106] P. L. Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *J. Comput. Phys.*, 43(2):357–372, 1981.

[107] Charles S and Peskin. Numerical analysis of blood flow in the heart. *J. Comput. Phys.*, 27(3):220 – 252, 1977.

[108] Youcef Saad and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[109] Yousef Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, 1996.

[110] Stefan Schirra. Robustness and precision issues in geometric computation. In J. R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry.* North Holland, 1999.

[111] Susanne Schmitt. The diamond operator for real algebraic numbers. ECG Technical Report ECG-TR-243107-01, Effective Computational Geometry for Curves and Surfaces, 2003.

[112] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. Technical report, Carnegie Mellon University, 1996.

[113] Charles Smith. *An Elementary Treatise on Conic Sections.* Macmillan, New York, 1893.

[114] Alvise Sommariva and Marco Vianello. Gauss-Green cubature and moment computation over arbitrary geometries. *J. Comput. Appl. Math.*, 231:886–896, 2009.

[115] Philippe R. Spalart and Steven R. Allmaras. A one-equation turbulence model for aerodynamics flows. AIAA 1992-0439, January 1992.

[116] Endre Süli and Paul Houston. Adaptive finite element approximation of hyperbolic problems. In T.J. Barth, M. Griebel, D. E. Keyes, R. M. Nieminen, D. Roose, and T. Schlick, editors, *Lecture Notes in Computational Science and Engineering: Error Estimation and Adaptive Discretization Methods in Computational Fluid Dynamics*, volume 25. Springer, Berlin, 2002.

[117] M. A. Taylor, B. A. Wingate, and R. E. Vincent. An algorithm for computing Fekete points in the triangles. *SIAM J. Numer. Anal.*, 38(5):1707–1720, 2000.

[118] V. Tchakaloff. Formules de cubatures méchaniques à coefficients non negatifs. *Bull. Sci. Math.*, 81:123–134, 1957.

[119] Lloyd N. Trefethen. Is Gauss quadrature better than Clenshaw–Curtis? *SIAM Rev.*, 50(1):67–87, 2008.

[120] John C. Vassberg, Mark A. DeHaan, and Tony J. Sclafani. Grid generation requirements for accurate drag predictions based on OVERFLOW calculations. AIAA 2003-4124, 2003.

[121] D. A. Venditti and D. L. Darmofal. Anisotropic grid adaptation for functional outputs: Application to two-dimensional viscous flows. *J. Comput. Phys.*, 187(1):22–46, 2003.

[122] M. Wheeler. An elliptic collocation-finite element method with interior penalties. *SIAM J. Numer. Anal.*, 15:152–161, 1978.

[123] M Wayne Wilson. Discrete least squares and quadrature formulas. *Math. Comp.*, 24(110):271–282, 1970.

[124] Masayuki Yano. *An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes.* PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2012.

[125] Masayuki Yano and David Darmofal. On dual-weighted residual error estimates for p-dependent discretizations. ACDL Report TR-11-1, Massachusetts Institute of Technology, 2011.

[126] Masayuki Yano and David L. Darmofal. Case C1.3: Flow over the NACA 0012 airfoil: Subsonic inviscid, transonic inviscid, and subsonic laminar flows. First international workshop on high-order CFD methods, 2012.

[127] Masayuki Yano and David L. Darmofal. An optimization-based framework for anisotropic simplex mesh adaptation. *J. Comput. Phys.*, 231(22):7626–7649, September 2012.

[128] Masayuki Yano, James M. Modisette, and David Darmofal. The importance of mesh adaptation for higher-order discretizations of aerodynamic flows. AIAA 2011–3852, June 2011.

[129] Chee K. Yap. Towards exact geometric computation. *Comput. Geom.*, 7(12):3 – 23, 1997.

[130] Chee Keng Yap. *Fundamental Problems of Algorithmic Algebra.* Oxford University Press, 1999.

[131] David P. Young, Robin G. Melvin, Michael B. Bieterman, Forrester T. Johnson, Satish S. Samant, and John E. Bussoletti. A higher-order boundary treatment for Cartesian-grid methods. *J. Comput. Phys.*, 92:1–66, 1991.

[132] Lingbo Zhang, Tao Cui, and Hui Liu. A set of symmetric quadrature rules on triangles and tetrahedra. *J. Comput. Math.*, 27(1):89–96, 2009.

[133] Xiaolin Zhong. A new high-order immersed interface method for solving elliptic equations with imbedded interface of discontinuity. *J. Comput. Phys.*, 225(1):1066 – 1099, 2007.

[134] Y.C. Zhou, Shan Zhao, Michael 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.