A Simplex Cut-Cell Adaptive Method for High-Order Discretizations of the Compressible Navier-Stokes Equations

by

Krzysztof Jakub Fidkowski

M.S., Aerospace Engineering (2004)
S.B., Aerospace Engineering (2003)
S.B., Physics (2003)
Massachusetts Institute of Technology

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

Doctor of Philosophy in Aerospace Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2007

Author . . . .
Department of Aeronautics and Astronautics
May 25, 2007

Certified by . . . .
David L. Darmofal
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by . . . .
Jaime Peraire
Professor of Aeronautics and Astronautics

Certified by . . . .
Per-Olof Persson
Instructor of Applied Mathematics

Accepted by . . . .
Jaime Peraire
Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

A Simplex Cut-Cell Adaptive Method for High-Order Discretizations of the Compressible Navier-Stokes Equations

by

Krzysztof Jakub Fidkowski

Submitted to the Department of Aeronautics and Astronautics
on May 25, 2007, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Aerospace Engineering

Abstract

While an indispensable tool in analysis and design applications, Computational Fluid Dynamics (CFD) is still plagued by insufficient automation and robustness in the geometry-to-solution process. This thesis presents two ideas for improving automation and robustness in CFD: output-based mesh adaptation for high-order discretizations and simplex, cut-cell mesh generation. First, output-based mesh adaptation consists of generating a sequence of meshes in an automated fashion with the goal of minimizing an estimate of the error in an engineering output. This technique is proposed as an alternative to current CFD practices in which error estimation and mesh generation are largely performed by experienced practitioners. Second, cut-cell mesh generation is a potentially more automated and robust technique compared to boundary-conforming mesh generation for complex, curved geometries. Cut-cell meshes are obtained by cutting a given geometry of interest out of a background mesh that need not conform to the geometry boundary. Specifically, this thesis develops the idea of simplex cut cells, in which the background mesh consists of triangles or tetrahedra that can be stretched in arbitrary directions to efficiently resolve boundary-layer and wake features.

The compressible Navier-Stokes equations in both two and three dimensions are discretized using the discontinuous Galerkin (DG) finite element method. An anisotropic h-adaptation technique is presented for high-order ($p > 1$) discretizations, driven by an output-error estimate obtained from the solution of an adjoint problem. In two and three dimensions, algorithms are presented for intersecting the geometry with the background mesh and for constructing the resulting cut cells. In addition, a quadrature technique is proposed for accurately integrating high-order functions on arbitrarily-shaped cut cells and cut faces. Accuracy on cut-cell meshes is demonstrated by comparing solutions to those on standard, boundary-conforming meshes. In two dimensions, robustness of the cut-cell, adaptive technique is successfully tested for highly-anisotropic boundary-layer meshes representative of practical high-Re simulations. In three dimensions, robustness of cut cells is demonstrated for various representative curved geometries. Adaptation results show that for all test cases considered, $p = 2$ and $p = 3$ discretizations meet desired error tolerances using fewer degrees of freedom than $p = 1$.

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

Acknowledgments

I would like to express my most sincere thanks to my advisor, Professor David Darmofal, for guiding me throughout the course of this thesis work. His guidance ranged from a spark of insight into a new problem to careful attention to detail at every step. He was accommodating in his advising style, letting me run with an idea when things were going well while taking the time to work through problems when the going got tough. In addition, I was fortunate enough to co-teach an aerodynamics class with him and to learn from his experience in effective teaching methods.

I am also grateful to my two other committee members, Professor Jaime Peraire and Dr. Per-Olof Persson. Their critical comments and feedback motivated several of the research directions considered in this work. Additionally, I would like to thank my readers, Professor Paul Houston, Dr. Venkat Venkatakrishnan, and Dr. Mori Mani, for providing very useful comments and suggestions on the thesis draft. Bob Haimes's comments on the 3D cut-cell chapter were also very helpful.

I'd like to thank the ever-evolving Project X team that has been part of my life for the last four years: Garrett and Todd, not only for contributing greatly to the code, but also for being great friends and putting up with my thesis excuses while co-teaching aerodynamics; Laslo for all his solver work that made many of the runs possible; David for his cut-cell visualization work; JM for using and debugging cut cells; Mike Park for useful discussions and for flying in for my defense; and all of the PX "has-beens": Matthieu, Paul, Mike, James, Doug, Eric, among others. In addition, Jean deserves recognition for being instrumental in making the entire lab run smoothly.

Outside of lab, a couple other groups have shaped my life as a graduate student. One of these is the MIT Triathlon Club, which has helped me stay somewhat in shape even in the most hectic of times. Nothing says camaraderie like swimming, biking, and running inordinate distances together. The other group is the Burton 2 residents, who have been a wonderful bunch over the past three years. You will be missed. The same goes for Roe and Bronwyn, the Burton-Conner housemasters – thank you for your dedication.

Finally, I'd like to thank my family. My parents, Zbigniew and Maria, and my parents-in-law, Jim and Nancy, for their continuous support. My siblings and siblings-in-law, Piotrek, Lukasz, Rob, and Liz for making life much less dull. Most of all, I'd like to thank my wife, Christina, for standing by me and making the last four years the best ones of my life.

This work was supported by the Department of Energy Computational Science Graduate Fellowship, under grant number DE-FG02-97ER25308.

Contents

List of Figures

Nomenclature

General

C_D	drag coefficient
C_f	skin friction coefficient
C_H	heat-transfer coefficient (Stanton number)
C_L	lift coefficient
d	dimension (i.e. 2 or 3)
$\mathbf{f}^v$	viscous shear force
M	Mach number
Ω	computational domain
Pe	Peclet number
$\mathbb{R}$	the set of real numbers
Re	Reynolds number

Discretization

$\delta^f_{ki}, \delta^{bf}_{ki}$	auxiliary variables used in the DG discretization of the viscous terms
F_{ki}	components of the inviscid flux; $K \times d$ values
F^v_{ki}	components of the viscous flux; $K \times d$ values
η^f, η^{bf}	stability factors used in the DG discretization of the viscous terms
κ	one finite element; this could be a triangle, a tetrahedron, or a cut cell
K	number of equations in the compressible Navier-Stokes system
$\mathbf{n}, n_i$	normal vector and components
p	solution approximation order; also used to denote pressure
q	for curved elements, order of geometric mapping from reference to physical space
$\mathcal{R}_H$	semi-linear weak form of the equations obtained from the discretization
T_H	set of all elements κ in a triangulation of the computational domain
$\mathbf{u}$, u_k	exact primal solution in $\mathcal{V}$ and components
$\mathbf{u}_H$, u_k	discrete primal solution in $\mathcal{V}_H$ and components
$\mathbf{v}_H$, v_k	test function in $\mathcal{V}_H$ and components
V^p_H	space of piecewise polynomials of order p over T_H
$\mathcal{V}$	infinite-dimensional solution space
$\mathcal{V}_H$	finite-dimensional solution space, equal to $[V^p_H]^K$
$\mathcal{W}_H$	infinite-dimensional solution space, equal to $\mathcal{V}_H + \mathcal{V}$

Error Estimation and Adaptation

$\mathbf{e}_i$	principal stretching directions for measuring anisotropy
e_0	user-requested global error level
$\tilde{e}_0$	modified requested error level, taking into account η_a and η_t
ϵ_κ	local error estimate/indicator on element κ; also used in adaptation to denote the expected local error estimate on the adapted mesh
ϵ_κ^c	in adaptation, current local error estimate/indicator on element κ
ϵ	global output error estimate equal to $\sum_\kappa \epsilon_\kappa$
$\mathbf{H}$	Hessian matrix $(d \times d)$ of second derivatives of a scalar quantity
h_i^c, h_i	current/desired principal stretching magnitudes
η_a	adaptation aggressiveness, $0 \le \eta_a < 1$
η_t	adaptation target $0 < \eta_t \le 1$
$\mathcal{J}$	output of interest, such as drag, lift, heat flux, etc.
$\mathbf{M}$	Riemannian metric defined for measuring lengths in the presence of anisotropy
n_κ	expected number of adapted-mesh elements contained in element κ of the current mesh
N_f	expected number of elements in the adapted mesh
$\boldsymbol{\psi}$	exact adjoint solution in $\mathcal{V}$
$\boldsymbol{\psi}_H$	discrete adjoint solution in $\mathcal{V}_H$
$\mathcal{P}_\kappa$	patch of elements neighboring κ (including κ)
r_κ	*a priori* convergence rate estimate for the error indicator on κ

Cut Cells

n_q	number of sampling points in cut-cell integration
$\mathbf{P}_j$	coefficient matrices (3×3) for quadratic Lagrange basis functions
$\Phi_{\mathbf{i}}(\mathbf{x})$	tensor-product Lagrange basis functions used for computing the $\zeta_{\mathbf{i}}(\mathbf{x})$
$\mathbf{R}$	$[X, Y, 1]^T$: vector of patch reference coordinates used with $\mathbf{P}_j$ and $\mathbf{S}_f$
$\mathbf{S}_f$	matrix representation (3×3) of a quadratic form
s	arc-length parameter used in splines and conics
$\mathbf{w}_q$	sampling weights associated with $\mathbf{x}_q$ for cut-cell integration
$\mathbf{X}$	$[X, Y]^T$: reference-space coordinates on quadratic patches
$\mathbf{x}, x_i$	physical-space coordinates. x, y, z are also used
$\mathbf{x}_q$	sampling points for cut-cell integration
$\zeta_{\mathbf{i}}(\mathbf{x})$	high-order basis functions onto which a cut-cell integrand is projected

Acronyms

CAD	Computer-Aided Design
CFD	Computational Fluid Dynamics
CFL	Courant-Freidrichs-Lewy number
DG	Discontinuous Galerkin
DOF	Degrees of Freedom
DPW	Drag Prediction Workshop
GMRES	Generalized Minimal Residual

Chapter 1

Introduction

1.1 Motivation

Over the last several decades, increased computational power coupled with improvements in numerical methods have made Computational Fluid Dynamics (CFD) an indispensable tool in analysis and design applications. In aerospace engineering, numerous CFD software packages exist that include sophisticated modeling and solution techniques. These CFD packages are used regularly in industry to reduce design cycle costs and to improve final product design. Wind-tunnel testing, often viewed as the alternative to CFD, is still performed, although it requires long turnaround times and is often restricted to a certain range of physical test conditions. On the other hand, the accessibility, relatively fast turnaround time, and almost arbitrary test conditions offered by CFD make it an attractive tool, especially for sensitivity studies, optimization, and preliminary vehicle design.

Given its prevalent use in industry, a natural question is whether CFD is a mature field. In particular, are the current CFD methods adequate for today's engineering purposes? As the following sections will show, recent evidence suggests that CFD is not yet a mature field. Understanding where improvements can be made requires a closer look at the typical CFD-based analysis process.

1.1.1 Use of CFD in Analysis

Typical use of CFD in analysis is illustrated in Figure 1-1. While this generalized view does not apply to all CFD methods, it holds for the most-commonly used finite volume and finite element methods. The starting point is a description of the geometry of interest, usually in the form of a Computer-Aided Design (CAD) model. For example, this geometry could be an airplane for which an engineer is interested in calculating forces under

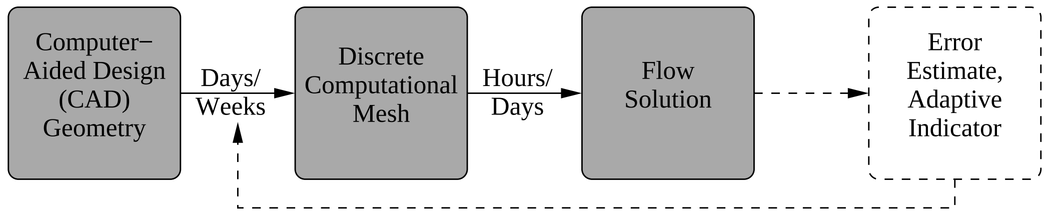

Computer−Aided Design (CAD) Geometry
Days/ Weeks
Discrete Computational Mesh
Hours/ Days
Flow Solution
Error Estimate, Adaptive Indicator

Figure 1-1: Typical use of CFD in analysis. Dashed arrow in the feedback direction refers to re-meshing of the computational domain based on an adaptive indicator.

certain prescribed flight conditions. Given the geometry, an engineer must create a discrete computational mesh of the flow domain, which in the example case is the volume outside the airplane. For complex geometries, construction of a "quality" mesh, one with adequate resolution of necessary features, can take days or even weeks. Once the mesh is constructed, a flow solution can be obtained within hours or days, depending on the size of the problem and on the computational resources available. The flow solution is then post-processed to calculate quantities of interest and sometimes visually examined to assess quality. In a design application, the geometry may be altered based on the post-processing and the cycle repeated.

Error estimation and adaptation is an additional step that is not often performed in practical CFD applications. As shown in Figure 1-1, this step occurs after the flow solution. It consists of estimating some measure of the error in the solution and providing an indicator of the areas in the approximate solution that contribute most to the error. This adaptive indicator can then be used to alter (e.g. refine) the computational mesh in an effort to minimize the error. Over several such adaptive iterations, the error could be reduced to a prescribed tolerance. The reasons that this step is rarely performed are that a reliable error indicator is rarely available and that adaptive re-meshing of a complex geometry is very time-consuming, lacks robustness, or is not even possible (especially for anisotropic meshes).

A notable example of CFD applied to a practical case is the Drag Prediction Workshop (DPW) run by the American Institute of Aeronautics and Astronautics (AIAA) [47, 42, 51]. One of the objectives of this workshop is "to assess the state-of-the-art computational methods as practical aerodynamic tools for aircraft force and moment prediction of industry-relevant geometries." This assessment is performed by providing the geometry of a standard aerodynamic computation to a variety of participating codes from industry, government, and academia and comparing the resulting force and moment predictions. The most recent workshop at the time of this writing was the DPW III. The geometry for this workshop consisted of a DLR-F6 wing-body configuration [15], shown in Figure 1-2. This geometry

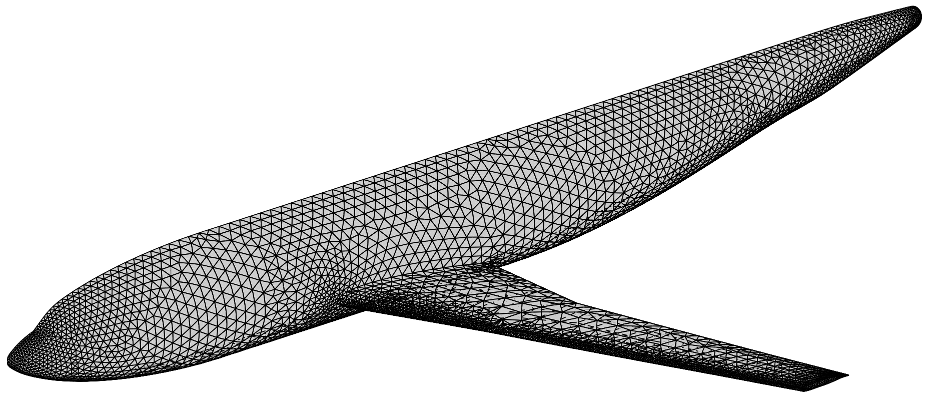


Figure 1-2: DLR-F6 wing-body geometry for the third AIAA Drag Prediction Workshop. The triangular surface mesh shown in this figure is used to define the quadratic patch geometry for the runs in Section 5.2.4.

was distributed to the participants, along with a number of structured and unstructured meshes, the finest meshes containing approximately 25 million elements. The resulting force and moment predictions for a fairly standard test case of $M = 0.7, C_L = 0.5, Re = 5 \times 10^6$ were collected. Figure 1-3 shows the total drag predictions obtained from the various codes on the finest meshes with the code index along the horizontal axis. Even after neglecting outlying data points, the spread in drag predictions is over 30 drag counts, where 1 drag count $= 10^{-4}$ of the drag coefficient, C_D. This spread is quite significant in terms of engineering accuracy: a simple range-equation analysis shows that for a typical large, long-range, passenger jet, a difference of 1 drag count translates into approximately 4-8 passengers, depending on whether the configuration is limited by fuel volume or weight [71, 24]. Clearly, for such an application, an uncertainty on the order of 30 drag counts is unacceptable for engineering analysis and design.

The results from the most recent workshop constitute only a slight improvement over the results from the two previous workshops [42, 47], even though computational power has increased substantially. This observation suggests that increases in computational power alone will be insufficient to decrease this uncertainty to acceptable levels in the near future. While part of the scatter can be attributed to different discretizations (e.g. cell-centered versus node-centered finite volume) and turbulence models (e.g. Spalart-Allmaras versus $k-\omega$), recent evidence points to differences in mesh size distribution as one of the dominant sources of the scatter [51].

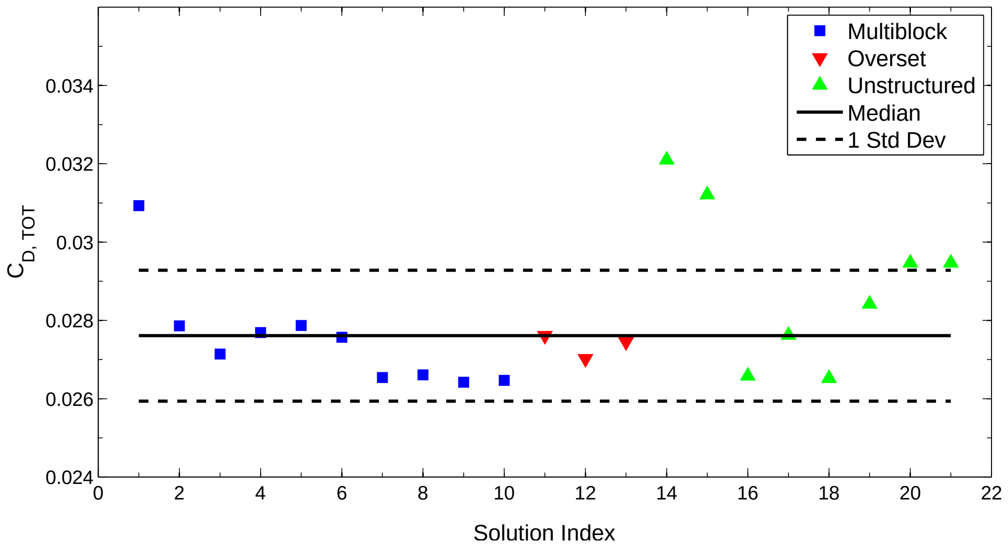


Figure 1-3: DPW III results [28, 53]: total drag coefficient predictions for the DLR-F6 wing-body at $M = 0.75$, $C_L = 0.5$, $Re = 5 \times 10^6$. The solution index differentiates between different codes, turbulence models, and mesh types.

1.1.2 Improving Robustness and Automation of CFD

The recent DPW results demonstrate that the risk of unacceptably large errors is high for current CFD practices. Typically, such risks are managed by practitioners who are knowledgeable about the assumptions and limitations of the models. However, even very experienced users cannot quantify the error in a discrete approximation of a complex flow-field. As a result, current CFD practices are not robust across the wide variety of existing applications, including ones such as the DPW case, for which many of the codes are tuned.

Lack of automation is another key issue that plagues the CFD analysis process. Current industry practices require heavy "person-in-the-loop" involvement, especially during mesh generation. As indicated in Figure 1-1, mesh generation may require days or weeks of user involvement. As such, this step is often the bottleneck in CFD analysis. This meshing bottleneck not only extends the design cycle time but also hinders the application of mesh adaptation methods and design optimization. Removing the user completely out of the design loop is neither possible nor advisable; however, improving automation in areas such as meshing is expected to reduce design cycle time and to allow for techniques such as solution-based adaptation and optimization.

The objective of this thesis is to demonstrate how current CFD practices can be improved to increase the robustness and automation of CFD in analysis and design. Two key ideas are suggested to demonstrate this objective: output-based error estimation and adaptation and a cut-cell meshing technique. With computational efficiency also in mind, these ideas will be presented in the context of a high-order discretization. The motivation and background for these ideas are presented in the following section.

1.2 Background

As discussed in the previous section, the proposed improvements to the automation and robustness of current CFD practices rely on two key ideas: output-based error estimation and adaptation and a simplex cut-cell meshing strategy. This section presents motivation and a review of the pertinent background for both of these ideas as well as for a high-order finite element discretization to which these ideas will be applied. Additional details are also provided in the respective chapters.

1.2.1 High-Order Methods

A high-order discretization enables practical computations at strict engineering-required error tolerances. In the context of this work, a high-order method is one with solution interpolation order, p, greater than 1. The benefit of using high order is motivated by estimating the time to solution for a high-fidelity CFD calculation. Assuming a solution error norm that converges at a rate $E = O(h^{p+1})$, where h is a measure of the mesh size, the time to solution, T, can be expressed as

$$\log T = d\left(-\frac{1}{p+1}\log E + a\log(p+1)\right) - \log F + \text{constant}.$$

In the above equation, d is the dimension, F is the computational speed, and a is the complexity of the solution algorithm. For example, $a = 2$ if calculations are dominated by dense matrix-vector multiplications. The derivation of this expression is outlined in [24]. When the accuracy requirement is high ($E << 1$) and a is moderate, the $\log E$ term will in general dominate the $\log(p+1)$ term; hence, the solution time will depend exponentially on $d/(p+1)$. In such high-fidelity calculations, increasing the order can significantly decrease the time to solution, or, alternatively, it can allow for solution of problems of much greater complexity.

Unfortunately, high-order discretizations are not prevalent in current CFD work in aerospace engineering. Finite volume discretizations have been the workhorse of CFD in aerospace engineering for the last couple decades. Although solution acceleration techniques and increased computational power have made large-scale computations practical, the spatial accuracy in current industry applications of finite volume are limited to, at best, second order. This means that the solution error, measured in some appropriate norm, decreases as h^r, $r \leq 2$, where h is a measure of grid spacing. Introducing high order in finite volume discretizations requires extended stencils, in which degrees of freedom become coupled beyond nearest-neighbor volumes. These extended stencils contribute to difficulties in stable iterative algorithms, memory requirements, and boundary conditions [24, 50]. On the other hand, finite element formulations introduce high-order degrees of freedom locally in each element and therefore yield an element-wise compact stencil.

The discontinuous Galerkin (DG) method is an example of a high-order finite element method in which element-to-element coupling exists only through fluxes on common boundaries. In particular, in DG, piecewise polynomials of arbitrary order are used to approximate the solution on each element, but solution continuity is not enforced at element interfaces. DG methods for hyperbolic conservation laws have been studied extensively in the literature [5, 6, 7, 11, 14, 19, 38]. These studies have demonstrated the realizability of high-order accuracy, error estimation, *hp*-adaptation, and stable discretization of the Euler and Navier-Stokes equations. This work uses a high-order DG discretization, the details of which are given in Chapter 2.

1.2.2 Error Estimation and Adaptation

Error estimation is vital to the usefulness of CFD. A CFD answer without an accompanying error estimate can compromise the fidelity of the analysis. Current practice of tuning CFD to certain representative on-design cases comes with no guarantees for other on-design configurations, much less for off-design cases or for novel geometries. Furthermore, meeting the mesh-size requirements is generally a user-intensive process and requires *a priori* experience in determining the locations of wakes, shocks, and other features.

Systematic error-estimation increases robustness of CFD by quantifying the solution error. In particular, an output-based error estimator ensures that outputs obtained from the solution are only used to the limits of their accuracy. Moreover, in conjunction with adaptation, error estimation closes the loop in the CFD analysis process depicted in Figure 1-1. This feedback in the loop yields an automated, adaptive method for controlling the solution error.

Error Estimation

The error in the solution can be quantified by various means. Discretization error is the difference between the calculated approximate solution and the exact solution. It is a function of location within the computational domain, although it can be integrated under a chosen norm over the entire domain to yield a global error or over individual elements to yield a local error. As the exact solution is unknown, the discretization error must be estimated; often this is done using a solution reconstruction process such as the one that will be described in Chapter 3. Another error estimate relies on the residual, which is obtained by substituting the approximate solution into the underlying partial differential equation. Nonzero residuals, calculated point-wise or in a weak sense on an enriched space, indicate regions where the governing equations are not strongly enforced. The residual can also be integrated to yield a global or element-wise local error estimate.

Zhang *et al* present adaptive results using discretization error and residual indicators for the Euler equations [79]. For one-dimensional, subsonic flows, Zhang *et al* find that a residual indicator is more efficient compared to a discretization-error indicator in driving the adaptation to reduce the total solution error. However, for transonic or multi-dimensional flows, neither indicator is adequately effective. In general, error estimates based on residual or discretization errors fail to capture propagation effects inherent to hyperbolic problems [39]. For hyperbolic problems, the residual and discretization error may not necessarily be large in certain crucial areas that significantly affect the solution downstream. For example, for separated flow over an airfoil, small perturbations in certain upstream areas may have large effects on the location of the separation point, which in turn has a large effect on the calculated lift and drag. Stated another way, engineering outputs can be highly sensitive to discretization or residual errors in areas that may not be easily identifiable *a priori*.

Fortunately, another type of error estimate, which is based on engineering outputs, addresses these problems. An engineering output is a quantity of interest for design purposes, such as the lift or drag on an airfoil. Techniques exist for estimating errors in engineering outputs. These techniques identify all areas of the domain that are important for the accurate prediction of an output, properly accounting for propagation effects in the process. A common output error estimation technique requires solution of an adjoint problem associated with the output, where the adjoint links local residuals to the output error. The resulting error estimate can be used to ascribe confidence levels to the engineering output or to drive an adaptive method with the goal of reducing the output error below a user-specified tolerance. This output-error estimation technique is employed in the current work. Section 3.1 presents further background and details.

Mesh Adaptation

One of the uses of error estimation is to drive an adaptive method that modifies the solution space in an attempt to decrease and equidistribute the error. For high-order finite element methods, in which degrees of freedom vary with the number of elements and with the interpolation order, the adaptation strategy can in general be classified into one of three categories: p-adaptation, h-adaptation, or hp-adaptation.

In p-adaptation, introduced by Szabo [70], the number of degrees of freedom is varied by changing the order of interpolation. With the discontinuous Galerkin method, changing the order is simple and can be done locally on each element. A recent example of p-adaptation applied to DG is given by Lu [48], who used an output-based error estimator to drive the adaptation. An advantage of p-adaptation is that the computational mesh remains fixed. In addition, an exponential error convergence rate with respect to degrees of freedom (DOF), $E \sim C_1^{(\mathrm{DOF})^{C_2}}$, is possible for sufficiently-smooth solutions. Disadvantages, however, include difficulty in handling singularities and areas of anisotropy and the need for a reasonable starting mesh.

In h-adaptation, the solution space is modified by adjusting the size of the elements in the computational mesh. Elements can be made smaller (refinement) or larger (coarsening), resulting in a local increase or decrease in the degrees of freedom. Mesh changes can be introduced locally, by splitting edges or adding extra nodes, or globally, by re-meshing the entire domain. A key feature of h-adaptation is that it allows for the generation of anisotropic (stretched) elements, which increase mesh efficiency in areas such as boundary layers and wakes. However, the best attainable error convergence is only algebraic with respect to DOF, $E \sim \mathrm{DOF}^{C_1}$.

hp-adaptation strives to combine the best of both strategies, employing p-refinement in areas where the solution is smooth and h-refinement near singularities or areas of anisotropy. The motivation for this strategy is that, in smooth regions, p-refinement is more effective at reducing the error per unit cost, compared to h-refinement [75, 40]. Implemented properly, hp-adaptation can isolate singularities and yield exponential error convergence with respect to DOF. In practice, however, the difficulty of hp-adaptation lies in making the decision between h- and p-refinement, a decision that requires either a solution regularity estimate or a heuristic algorithm. Houston and Süli [40] present a review of commonly used methods for making this decision.

The adaptation strategy chosen for this work is h-adaptation at a constant p. This strategy does not take advantage of the cost savings offered by hp-adaptation, but it avoids the additional complexity involved in making the regularity estimation decision. This simplification also allows for a straightforward comparison of the adaptive performance of different interpolation orders. Extension to hp-adaptation is one of the areas of possible future work.

1.2.3 Cut Cells

Currently, most industry-level meshers employ multiblock or fully-unstructured mesh-generation techniques. Multiblock mesh generation consists of subdividing the computational domain into block volumes for which structured meshes are easier to generate. The user generally has control of the number, size, location, and refinement level of the blocks, enabling targeted resolution of areas that are known *a priori* to require significant refinement. However, complex geometries usually require non-trivial multiblock subdivisions that result in significant user involvement in the mesh-generation process. A common alternative to multiblock mesh generation is unstructured mesh generation, in which the mesh connectivity is explicitly stored. Typically, unstructured meshes consist of triangles or tetrahedra. While unstructured meshers are often more automated, they generally offer less user control of sizing and suffer from robustness problems for stretched meshes around complex geometries.

One option for more automated and robust meshing is the use of cut cells, in which the computational mesh is cut out from a background mesh that need not conform to the geometry of interest. Without the boundary-conforming constraint, generation of the background mesh is straightforward and can be incorporated into an adaptive solution process. The burden of robustness is transferred to intersecting the background mesh with the geometry, a process that can be fully-automated.

Current finite volume/finite element computational meshes fall into one of the following categories: structured, boundary-conforming; unstructured, boundary-conforming; Cartesian, cut-cell. Strictly speaking, structured meshes are those for which the mesh connectivity is not stored, but rather implied in the ordering of nodes or elements. Often, structured meshes consist of rectangles in 2D and boxes in 3D, although this need not be the case. Cartesian meshes consist of rectangles or boxes, but, depending on how they are refined, need not be strictly structured.

Structured meshes have the advantage that associated solution methods are often memory-lean and fast. However, generation of boundary-conforming structured meshes on arbitrary geometries is not automated and requires significant user involvement. Unstructured meshes can often be generated automatically for geometries that are not overly complex and for linear geometry approximations. However, curved meshes have been found necessary for certain high-order methods, such as boundary-conforming DG discretizations [5]. Currently, construction of curved meshes for practical configurations is neither automated nor robust. One of the difficulties is ensuring that a curved geometry boundary does not intersect any interior faces, as shown for 2D in Figure 1-4. This is a difficult task for highly-anisotropic boundary layer meshes, in which several layers of interior faces may intersect the curved boundary . In addition, even linear, unstructured mesh generation is not bulletproof for very complex geometries. Compared to their structured counterparts, unstructured meshes are not as lean since they have to store mesh connectivity.

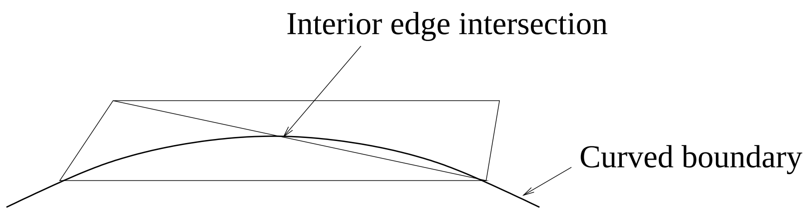


Figure 1-4: Example of a curved boundary intersecting an interior edge adjacent to two anisotropic triangles. Attempting to curve the boundary edge introduces a negative Jacobian in the mapping from the reference triangle to the curved element and hence renders the triangulation invalid.

The Cartesian Method

The "Cartesian method" is a meshing technique in which rectangular/hexahedral cells on a regular lattice are allowed to cut through the geometry, resulting in "cut cells" on the geometry boundary, as shown in Figure 1-5. Mesh adaptation is in general necessary to resolve the boundary well. Since the boundary-conforming constraint is removed, the Cartesian mesh generation process can be fully-automated. The costs of this automation are the additional required capability of intersecting the geometry with a background mesh and the ability to use arbitrarily-shaped cut cells in the flow solver. However, given the large cost of boundary-conforming mesh generation, the automation benefit of cut cells may be worth the additional effort.

The idea of using Cartesian cut cells began with the works of Purvis and Burkhalter in 1979 [62] and Wedan and South in 1983 [76]. These authors worked with a finite volume method for the full potential equations in which the geometry was cut out in a piecewise linear fashion on each cell. This work was extended to the 2D Euler equations by Clarke, Salas, and Hassan in 1986 [18], who also added an agglomeration technique in which small cells were incorporated into adjacent cells so as not to limit the allowable time step. Their work showed reasonable agreement with an analytical airfoil solution, except at the leading edge, where the grid was deemed too coarse. Shortly thereafter, Gaffney, Salas, and Hassan [29] extended the Euler finite volume method to 3D, still using linear cuts and small volume cut-cell agglomeration. They found that when the geometry surfaces were not grid-aligned, heavy (isotropic) clustering was required to sufficiently resolve the flow.

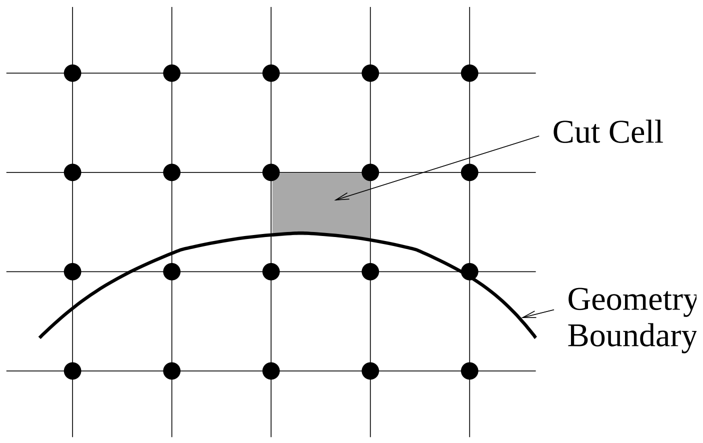


Figure 1-5: Sample Cartesian mesh in two dimensions. The square lattice mesh does not conform to the geometry. Cut cells are portions of intersected elements that lie inside the computational domain (above the geometry boundary in this case).

Around the same time, a group at Boeing developed a Cartesian cut-cell method for the 3D potential flow equations on complex geometries. The Cartesian method was chosen because, while geometries were at hand from previous linear panel codes, robust techniques for volume mesh generation around these geometries were not available. Rubbert *et al* [66] and Young *et al* [78] presented details of the resulting Cartesian cut-cell finite element method, which became the TRANAIR code. The method is based on the construction of a conforming finite element basis on linear cut cells, using Stokes' theorem to carry out the volume integration. The method also allows for adaptation based on geometry (length scale of panels), solution features, and user-prescribed refinement. Since its inception, TRANAIR has undergone several upgrades and is still in active use. Its success is primarily due to the robustness and automation inherent in the cut-cell mesh generation technique.

In the late 1980's, Leveque looked into relaxing the time-step limit imposed on small cut cells frequently encountered in the Cartesian finite volume method [45, 46]. His resulting generalized Godunov method accounted for wave propagation through more than one cell, and he was able to implement the method in two dimensions. Berger and Leveque [10] then presented a 2D Cartesian mesh method that incorporated the time step fix and an isotropic adaptation technique based on Richardson extrapolation. In this work, they noted that general anisotropic grid stretching would be a formidable challenge for the Cartesian method.

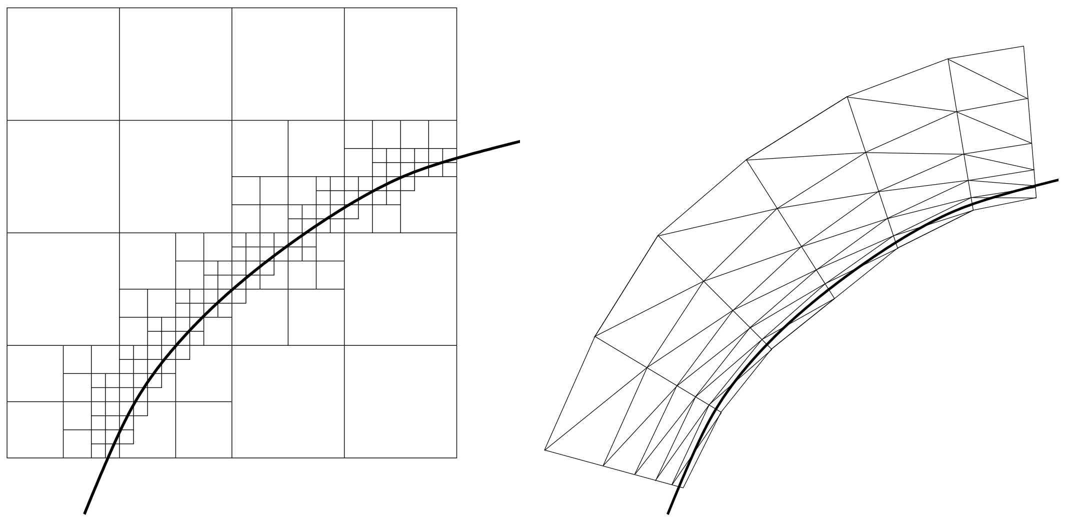
In the early 1990's, the Cartesian method for finite volume gained popularity. Quirk [63] used Bezier curves for 2D geometry definition, although he still only allowed linear cuts, and an adaptive mesh refinement technique similar to that of Berger and Leveque [10]. De Zeeuw and Powell [22] presented a 2D Euler Cartesian method that incorporated adaptation on solution gradients and a local time stepping procedure. Melton *et al* [52] presented a 3D Euler Cartesian method with an automated cutting algorithm using CAD-based surface triangulation intersections and local geometry-based grid refinement. Pember and Bell [59] also worked with a 3D Euler Cartesian method but allowed for solution-based adaptation using Richardson extrapolation.

Extending the Cartesian method from Euler to Navier-Stokes entails two challenges. First, at least for finite volume, accurate treatment of the viscous flux terms is difficult on irregularly-shaped cut cells. Second, anisotropic adaptation is not possible in general, non-grid aligned directions. Nevertheless, Coirier and Powell [20] applied the Cartesian method "as-is" to the 2D Navier-Stokes equations. With a diamond-path reconstruction scheme for the viscous term and isotropic adaptation, they obtained good results but mentioned that isotropic adaptation would become prohibitive in 3D. Karman [41] undertook the solution of the 3D Reynolds-averaged Navier-Stokes (RANS) equations. His resulting code, SPLITFLOW, takes as input a geometry together with an anisotropic prismatic boundary-layer mesh, and generates a Cartesian grid that intersects the outer portion of the boundary-layer mesh. Karman was able to obtain results for complex geometries, but his technique requires user construction of a viscous grid, which in turn requires *a priori* knowledge of the position, extent, and necessary refinement of the boundary layers and wakes. Such a requirement hinders the automation and robustness of the resulting method.

The late 1990's saw more work on the Cartesian method with researchers bolstering strengths such as automated mesh generation and tackling outstanding issues such as anisotropy and small cut cells. Lahur *et al* [43, 44] looked into anisotropic splitting of Cartesian meshes using horizontal or vertical refinement. Such adaptation resulted in savings only for grid-aligned features. Leveque continued working on high-resolution wave propagation in finite volume and introduced the CLAWPACK software package. This package was used subsequently by Forrer and Jeltsch [27], who gave a boundary treatment based on reflecting flowfield at a straight boundary line, and Calhoun and Leveque [16], who considered the advection-diffusion problem using a capacity function. In 1997, Aftosmis presented a comprehensive review of the Cartesian method that focused on geometric algorithms and surface modeling [1]. One of Aftosmis's takeaway messages is that an important advantage of the Cartesian method is separating the geometry mesh from the solution mesh. He also presented a counting argument demonstrating why anisotropic adaptation is crucial for 3D. In addition, Aftosmis mentioned that Cartesian methods often store full grid connectivity anyway, due to adaptation, resulting in so-called "unstructured Cartesian" approaches. Aftosmis *et al* [2] then presented the details of a 3D Cartesian solver package, Cart3d, that featured fast and automated mesh generation using surface geometry triangulation intersections. Cart3d is currently in use for large scale computations, including space shuttle ascent debris calculations [55]. Ongoing work continues in computing adjoints and shape sensitivities [57] and in novel ways of moving beyond Euler calculations [3]. However, it appears that a practical viscous discretization for the Cartesian method is going to be a tough challenge to overcome.

Cut Cells on Simplex Elements

The Cartesian method offers a robust and automated alternative to boundary-conforming mesh generation with advantages realized primarily for complex geometries. However, the use of a regular lattice in one Cartesian coordinate system precludes the possibility of anisotropic mesh adaptation along directions not aligned with the grid, as illustrated in Figure 1-6a. Shown in the figure is a mesh of a boundary layer with a certain minimum required mesh size in a direction normal to the boundary. While the mesh size requirement in the streamwise direction along the boundary is much less stringent, the Cartesian refinement mechanism cannot capture this anisotropy. This lack of practical anisotropic adaptation is a major obstacle in applying the Cartesian method to the Navier-Stokes or RANS equations.



(a) Cartesian mesh

(b) Triangular mesh

Figure 1-6: Comparison of Cartesian and triangular cut-cell meshes of a curved boundary layer. As the boundary is not aligned with the grid, isotropic refinement is required for the Cartesian mesh (a). With triangular cut cells, anisotropic refinement is possible in general directions (b).

The need for anisotropic adaptation motivates another cut-cell mesh generation technique: simplex cut cells. Simplex elements are triangles in two dimensions and tetrahedra in three dimensions. Figure 1-6b shows a triangular mesh of the same boundary-layer flow as in Figure 1-6a. Without a regular lattice, arbitrarily-shaped elements are possible. In particular, the anisotropy of the boundary layer is reflected in the mesh, which contains fewer elements for the same resolution. Of course, this method shares the drawback of any unstructured method: the mesh connectivity has to be stored. However, for practical, viscous simulations, the gains of general anisotropic adaptation are likely to outweigh this cost.

The mechanics of the simplex cut-cell method introduced in this work can be extended to other element shapes. Simplices were chosen because automated, metric-driven meshers exist for generating triangular and tetrahedral elements. These meshers are robust when the boundary-conforming requirement is removed. That is, the mesh generation problem reduces to creating a mesh for a simple shape such as a box with boundaries at the farfield. An important aspect of applying cut cells to a high-order finite element method is dealing with curved boundaries and integration on the interiors of arbitrarily-shaped elements. These topics will be addressed in Chapters 4 and 5.

1.3 Thesis Overview

This thesis addresses the development of a robust adaptation methodology for high-order discretizations, focusing on all aspects of the adaptation process. These aspects include output-based error estimation, anisotropic mesh adaptation, and simplex cut-cell meshing. The specific contributions of this thesis are as follows:

- Extension of solution anisotropy detection from $p = 1$ to higher-order interpolation.
- Goal-oriented mesh optimization that incorporates predictions of the adapted mesh size during error equidistribution.
- Triangular cut-cell meshing and associated intersection with curved spline geometries.
- Tetrahedral cut-cell meshing and associated intersection with curved quadratic-patch surface representations.
- A sampling-point-based integration technique for arbitrarily-shaped volumes and areas in two and three dimensions.

While these contributions are intended to be general, this work applies the methods developed to a discontinuous Galerkin finite element discretization of the compressible Navier-Stokes equations. Details of the discretization are given in Chapter 2. Chapter 3 outlines the output-based error estimation procedure and the anisotropic adaptation strategy. Special attention is given to anisotropic adaptation for high-order interpolation and to a simple, yet efficient, mesh optimization algorithm. Chapters 4 and 5 describe the details of simplex cut cells in two and three dimensions, respectively. Both chapters contain results demonstrating the accuracy of cut cells compared to boundary conforming meshes and their performance in the output-based adaptive method. Finally, conclusions and ideas for future work are given in Chapter 6.

Chapter 2

Compressible Navier-Stokes Discretization

While the error estimation, adaptation, and cut-cell methods to be presented are valid for general equations, the target application for this work is the compressible Navier-Stokes equations. For completeness, this chapter presents the Navier-Stokes equations and their discretization via the discontinuous Galerkin (DG) finite element method. This method is introduced in the first section, using an advection example.

2.1 Discontinuous Galerkin Example

This section illustrates the basic features of the DG method applied to the scalar advection equation. Using index notation with implied summation, the advection equation reads

$$\partial_i F_i(u) = 0, \qquad F_i(u) = V_i u, \tag{2.1}$$

where V_i are components of a prescribed velocity field, u is a scalar quantity, and $i \in [1, .., d]$ indexes the spatial dimension, d. (2.1) is a conservation statement for u when u advects with velocity $\mathbf{V} = [V_i]$. A standard finite element discretization proceeds by triangulating the computational domain, Ω, into elements κ and searching for a solution, u_H, in a finite-dimensional space, V_H^p, for which a weak form of (2.1) is satisfied. V_H^p is the space of piecewise polynomials of order p over the elements. Figure 2-1 illustrates a sample solution $u_H \in V_H^p$ over two elements. Note, T_H refers to the set of elements in the triangulation. As shown, V_H^p admits discontinuities across the elements, allowing for greater freedom in the choice of basis functions on each element compared to the continuous finite element

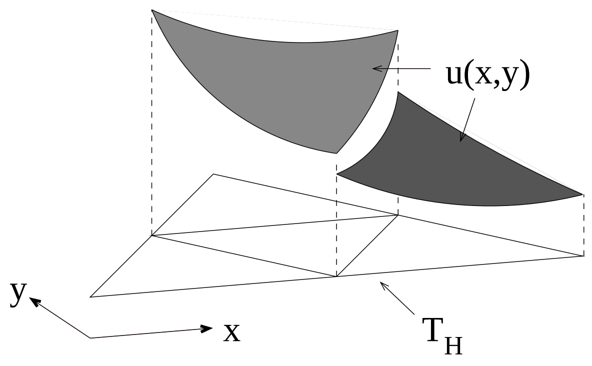


Figure 2-1: Sample solution u_H in V_H^p, the space of piecewise continuous polynomials of order p. u_H is shown over two elements in a two-dimensional mesh.

method. Specifically, the same solution space can be used for arbitrarily-shaped cut elements regardless of the number and location of adjacent elements.

A weighted residual statement, or weak form, is obtained by multiplying (2.1) by test functions $v_H \in V_H^p$ and integrating over the elements. Considering one element, κ, the weak form is obtained by an integration by parts,

$$\begin{aligned} \int_\kappa \partial_i F_i(u_H) v_H d\mathbf{x} &= 0, \\ -\int_\kappa F_i(u_H)\partial_i v_H, d\mathbf{x} + \int_{\partial\kappa} \widehat{F}_i(u_H^+, u_H^-) n_i v_H^+ ds &= 0. \end{aligned} \tag{2.2}$$

The n_i are components of the outward-pointing normal vector, and the notation $()^+$ and $()^-$ refers to quantities taken from the interior and exterior of κ, respectively. $\widehat{F}_i(u_H^+, u_H^-)$ is a suitably-chosen average flux on the boundary of κ, where u_H may be discontinuous. For example, for the advective flux in (2.1), a suitable choice for $\widehat{F}_i(u_H^+, u_H^-)$ is full upwinding,

$$\widehat{F}_i(u_H^+, u_H^-) n_i = \frac{1}{2} V_i n_i \left(u_H^- + u_H^+\right) - \frac{1}{2} |V_i n_i| \left(u_H^- - u_H^+\right).$$

Summing (2.2) over all elements yields the desired weak form, $\mathcal{R}_H(u_H, v_H)$. With a suitably chosen basis for V_H^p, this weak form becomes a system of equations, which can be solved to yield u_H.

2.2 Compressible Navier-Stokes Equations

The compressible Navier-Stokes system consists of K equations, where $K = d + 2$ for laminar flow in d dimensions. The first equation is a statement of conservation of mass, the next d equations represent conservation of momentum, and the final equation represents conservation of energy. The k^{th} equation, written using index notation, reads

$$\partial_t u_k + \partial_i F_{ki}(\mathbf{u}) - \partial_i F^v_{ki}(\mathbf{u}) = 0, \tag{2.3}$$

where $i \in [1, .., d]$ indexes the spatial dimension, and $\mathbf{u}$ is the state vector with K components, u_k. In this work, the conservative state vector is used, $\mathbf{u} = [\rho, \rho v_i, \rho E]$, where ρ is the density, v_i are the d components of the velocity, and E is the total energy. $F_{ki}(\mathbf{u})$ and $F^v_{ki}(\mathbf{u})$ are inviscid and viscous flux components, respectively, chosen such that (2.3) is a compact expression for the conservation of mass, momentum, and energy. These flux components are:

Conservation of Mass, $k = 1$:

$$F_{1i} = \rho v_i, \qquad F^v_{1i} = 0.$$

Conservation of Momentum, $k = 2, .., d+1$:

$$F_{ki} = \rho v_{k-1} v_i + \delta_{(k-1)i}\, p, \qquad F^v_{ki} = \tau_{(k-1)i}.$$

Conservation of Energy, $k = d+2$:

$$F_{(d+2)i} = \rho v_i H, \qquad F^v_{(d+2)i} = \kappa_T \partial_i T + v_j \tau_{ij}.$$

In the above equations, j indexes the spatial dimension, δ_{ij} is the Kronecker delta function, with $\delta_{ij} = 1$ if $i = j$, and 0 if $i \neq j$, and τ_{ij} are the viscous shear and normal stresses for a Newtonian fluid,

$$\tau_{ij} = \mu(\partial_i v_j + \partial_j v_i) + \delta_{ij} \lambda \partial_m v_m, \tag{2.4}$$

where m indexes the spatial dimension. The pressure, p, total enthalpy, H, and temperature, T, are related via

$$\begin{aligned} p &= (\gamma - 1)\left(\rho E - \frac{1}{2}\rho(v_i v_i)\right), \\ H &= E + p/\rho, \\ T &= \frac{p}{\rho R}. \end{aligned}$$

Relevant physical quantities for air are,

$$\begin{aligned}
\text{Dynamic viscosity:} \quad \mu &= \mu_{\text{ref}} \left(\frac{T}{T_{\text{ref}}}\right)^{1.5} \left(\frac{T_{\text{ref}} + T_{\text{s}}}{T + T_{\text{s}}}\right), \\
& \quad (\text{Sutherland's law: } T_{\text{ref}} = 288.15\text{K},\ T_{\text{s}} = 110\text{K}) \\
\text{Bulk viscosity coefficient:} \quad \lambda &= -\frac{2}{3}\mu, \\
\text{Thermal conductivity:} \quad \kappa_T &= \frac{\gamma \mu R}{(\gamma - 1)Pr}, \\
\text{Specific-heat ratio:} \quad \gamma &= 1.4, \\
\text{Prandtl number:} \quad Pr &= 0.71, \\
\text{Gas constant:} \quad R&.
\end{aligned}$$

The fluxes are nonlinear functions of the state vector components. It is convenient to make use of the linear dependence of F^v_{ki} on the spatial gradients $\partial_j u_l$ by writing

$$F^v_{ki} = A_{kilj}(\mathbf{u})\partial_j u_l, \tag{2.5}$$

where $A_{kilj}(\mathbf{u})$ is a tensor that is a nonlinear function of the state vector components and l indexes the state vector.

2.3 Discontinuous Galerkin Discretization

As in the advection example at the beginning of this chapter, the discretization of (2.3) proceeds in standard finite element fashion by triangulating the computational domain, Ω, into elements κ and searching for a solution, $\mathbf{u}_H$, in a finite-dimensional space, $\mathcal{V}_H$, for which a weak form of (2.3) is satisfied. As $\mathbf{u}_H$ is a state vector with K components, $\mathcal{V}_H = [V^p_H]^K$; that is, each component of $\mathbf{u}_H$ resides in V^p_H, the space of piecewise polynomials of order p over the elements.

The weak form is presented here for one element κ with boundary $\partial\kappa$. The steady-state, discrete semi-linear form, $\mathcal{R}_H(\mathbf{u}_H, \mathbf{v}_H)$, follows by summing over all elements,

$$\mathcal{R}_H(\mathbf{u}_H, \mathbf{v}_H) = \sum_{\kappa} \left(\mathbb{E}_\kappa(\mathbf{u}_H, \mathbf{v}_H) + \mathbb{V}_\kappa(\mathbf{u}_H, \mathbf{v}_H)\right) = 0, \tag{2.6}$$

where $\mathbb{E}_\kappa(\mathbf{u}_H, \mathbf{v}_H)$ is the contribution of the inviscid flux, $\mathbb{V}_\kappa(\mathbf{u}_H, \mathbf{v}_H)$ is the contribution of the viscous flux, and $\mathbf{v}_H \in \mathcal{V}_H$ denotes an arbitrary test function. In the equations that follow, v_k refers to components of $\mathbf{v}_H$, and u_k refers to components of $\mathbf{u}_H$. Of particular relevance to the cut-cell algorithm is the fact that construction of the residual requires element-interior area integrals in addition to element-boundary integrals.

First, $\mathbb{E}_\kappa(\mathbf{u}_H, \mathbf{v}_H)$ is obtained by forming the inner product of $\partial_i F_{ki}$ in (2.3) with the test-function components, v_k, and integrating by parts over the element. The resulting expression is

$$\mathbb{E}_\kappa(\mathbf{u}_H, \mathbf{v}_H) = -\int_\kappa \partial_i \mathrm{v}_k F_{ki} d\mathbf{x} + \int_{\partial\kappa} \mathrm{v}_k^+ \widehat{F}_{ki}(\mathbf{u}_H^+, \mathbf{u}_H^-) n_i ds,$$

where n_i is the outward-pointing normal, and $\widehat{F}_{ki}$ is an approximate characteristic-based flux function (Roe-averaged flux in this work [65]). Boundary conditions are imposed by setting $\widehat{F}_{ki}$ appropriately when $\partial\kappa$ is on the domain boundary, $\partial\Omega$, as described in Appendix A for the boundary conditions used in this work.

The viscous flux term contribution is discretized using the second form of Bassi & Rebay (BR2) [6]. In this form, the steady-state Navier-Stokes equations are re-written as a system of first-order equations by introducing Q_{ki},

$$\begin{aligned} \partial_i F_{ki} - \partial_i Q_{ki} &= 0, \\ Q_{ki} - A_{kilj} \partial_j \mathrm{u}_l &= 0. \end{aligned}$$

Taking the inner product of the first equation set with test function components $\mathrm{v}_k \in V_H^p$, and the second equation set with test function components $\mathrm{w}_{ki} \in V_H^p$, yields, after an integration by parts,

$$\mathbb{E}_\kappa(\mathbf{u}_H, \mathbf{v}_H) + \int_\kappa \partial_i \mathrm{v}_k Q_{ki} dx - \int_{\partial\kappa} \mathrm{v}_k^+ \widehat{Q}_{ki} n_i ds = 0, \tag{2.7}$$

$$\int_\kappa \mathrm{w}_{ki} Q_{ki} dx + \int_\kappa \partial_j \Big(\mathrm{w}_{ki} A_{kilj} \Big) \mathrm{u}_l dx - \int_{\partial\kappa} \mathrm{w}_{ki}^+ \widehat{A_{kilj}\mathrm{u}_l} n_j ds = 0, \tag{2.8}$$

where $\widehat{\cdot}$ denotes flux averaging for discontinuous quantities. Note, since $1 < k \leq K$ and $1 < i \leq d$, the w_{ki} are components of vector-valued functions of size Kd. Setting $\mathrm{w}_{ki} = \partial_i \mathrm{v}_k$ in (2.8), substituting for $\int_\kappa \partial_i \mathrm{v}_k Q_{ki} dx$ in (2.7), and integrating by parts one more time yields the viscous contribution to the weak form,

$$\begin{aligned} \mathbb{V}_\kappa(\mathbf{u}_H, \mathbf{v}_H) &= \int_\kappa \partial_i \mathrm{v}_k A_{kilj} \partial_j \mathrm{u}_l dx - \int_{\partial\kappa} \partial_i \mathrm{v}_k^+ \Big(A_{kilj}^+ \mathrm{u}_l^+ - \widehat{A_{kilj}\mathrm{u}_l} \Big) n_j ds \\ &\quad - \int_{\partial\kappa} \mathrm{v}_k^+ \widehat{Q}_{ki} n_i ds. \end{aligned}$$

The choice of $\widehat{Q}_{ki}$ and $\widehat{A_{kilj}\mathrm{u}_l}$ is not unique, but only certain choices produce discretizations that are consistent, dual-consistent, and compact [48]. The set of fluxes used in this work is shown in Table 2.1.

Table 2.1: Viscous fluxes

	$\widehat{Q}_{ki}$	$\widehat{A_{kilj}\mathrm{u}_l}$
Interior	$\{A_{kilj}\partial_j \mathrm{u}_l\} - \eta^f\{\delta^f_{ki}\}$	$A^+_{kilj}\{\mathrm{u}_l\}$
Boundary, Dirichlet	$A^b_{kilj}\partial_j \mathrm{u}^+_l - \eta^{bf}\delta^{bf}_{ki}$	$A^b_{kilj}\mathrm{u}^b_l$
Boundary, Neumann	$\Big(A_{kilj}\partial_j \mathrm{u}_l\Big)^b$	$A^+_{kilj}\mathrm{u}^+_l$

The operator $\{\cdot\}$ denotes the average across an element boundary, $\{\cdot\} = \frac{1}{2}\big((\cdot)^+ + (\cdot)^-\big)$, the superscript b indicates values computed using a state appropriately constructed from boundary conditions (Appendix A), and η^f and η^{bf} are constant stability factors set to 3 and 3/2, respectively. $\delta^f_{ki}, \delta^{bf}_{ki} \in V^p_H$ are components of auxiliary variables for interior and boundary faces, respectively, that satisfy, $\forall \mathrm{w}_{ki} \in V^p_H$,

$$\begin{aligned}
\int_{\kappa^+} \delta^{f+}_{ki}\mathrm{w}_{ki}dx + \int_{\kappa^-} \delta^{f-}_{ki}\mathrm{w}_{ki}dx &= \int_{\sigma^f} \{\mathrm{w}_{ki}A_{kilj}\}\Big(\mathrm{u}^+_l - \mathrm{u}^-_l\Big)n_j ds,\\
\int_{\kappa} \delta^{bf}_{ki}\mathrm{w}_{ki}dx &= \int_{\sigma^{bf}} \mathrm{w}_{ki}A^b_{kilj}\Big(\mathrm{u}^+_l - \mathrm{u}^b_l\Big)n_j ds,
\end{aligned}$$

where σ^f and σ^{bf} denote interior and boundary faces, respectively, and κ^+, κ^- are elements on either side of σ^f. This viscous discretization yields a compact stencil in that the element-to-element influence is only nearest-neighbor.

Chapter 3

Output-based Error Estimation and Adaptation

3.1 Output-based Error Estimation

Output-based error estimation and adaptation for CFD have been studied extensively in the literature [61, 9, 49, 31, 36, 4, 26, 74, 48]. In the following analysis, an output error estimate for a generic weighted residual statement is derived, motivated by the previously cited work. This estimate is then applied to the DG weighted residual statement. Before presenting the derivation of the output error estimate, however, the concept of an *adjoint* solution is introduced.

3.1.1 The Adjoint

As discussed in Section 1.2.2, accurate prediction of an output (e.g. drag or lift) may depend on resolution of seemingly un-interesting areas. This is especially the case in hyperbolic problems, in which disturbances do not necessarily decay away from the source but rather propagate and affect the solution downstream. Error estimators based on local criteria often fail to capture the error due to such propagation effects. Output-based error estimators address this problem by linking local residuals to outputs through the use of the adjoint solution.

The adjoint can be thought of as a Green's function that relates the residual of an underlying partial differential equation (PDE) to an output derived from the PDE solution. For a finite-dimensional problem, the adjoint vector weights the residual vector in an inner product that gives the change in the output. For example, let $\mathbf{u} \in \mathbb{R}^n$ be a solution to a nonlinear system of equations,

$$\mathbf{F}(\mathbf{u}) = 0, \qquad \mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n. \tag{3.1}$$

Given a source (i.e. residual), $\delta\mathbf{r} \in \mathbb{R}^n$, the solution is perturbed: $\mathbf{F}(\mathbf{u} + \delta\mathbf{u}) = \delta\mathbf{r}$. For infinitesimally-small sources and perturbations, $\mathbf{F}$ can be linearized about $\mathbf{u}$, resulting in a linear system,

$$\mathbf{F}'[\mathbf{u}]\delta\mathbf{u} = \delta\mathbf{r}, \tag{3.2}$$

where $\mathbf{F}'[\mathbf{u}] = \partial\mathbf{r}/\partial\mathbf{u}$ is the $n \times n$ Jacobian matrix. Let $\mathcal{J}(\mathbf{u})$ be an output of interest, where $\mathcal{J}$ may be a nonlinear function of $\mathbf{u}$. Evaluating the output with the perturbed solution results in a change of

$$\delta\mathcal{J} = \mathcal{J}(\mathbf{u} + \delta\mathbf{u}) - \mathcal{J}(\mathbf{u}) = \mathcal{J}'[\mathbf{u}]\delta\mathbf{u}, \tag{3.3}$$

where $\mathcal{J}'[\mathbf{u}] = \partial\mathcal{J}/\partial\mathbf{u}$ is a $1 \times n$ row vector of partial derivatives of $\mathcal{J}$ with respect to the entries of $\mathbf{u}$. By the chain rule,

$$\frac{\partial\mathcal{J}}{\partial\mathbf{u}} = \frac{\partial\mathcal{J}}{\partial\mathbf{r}}\frac{\partial\mathbf{r}}{\partial\mathbf{u}}, \tag{3.4}$$

where $\partial\mathcal{J}/\partial\mathbf{r}$ is a $1 \times n$ row vector. The adjoint vector, $\boldsymbol{\psi} \in \mathbb{R}^n$, associated with the output $\mathcal{J}$ is defined as the variation of the output with respect to the residual,

$$\boldsymbol{\psi} = \left(\frac{\partial\mathcal{J}}{\partial\mathbf{r}}\right)^T,$$

where the transpose makes $\boldsymbol{\psi}$ an $n \times 1$ column vector. Taking the transpose of (3.4), the adjoint is obtained from the solution of the following $n \times n$ linear system:

$$\mathbf{F}'[\mathbf{u}]^T\boldsymbol{\psi} \quad = \quad \mathcal{J}'[\mathbf{u}]^T. \tag{3.5}$$

Multiplying (3.5) by $\delta\mathbf{u}^T$ yields

$$\begin{aligned} \underbrace{\delta\mathbf{u}^T\mathbf{F}'[\mathbf{u}]^T}_{\delta\mathbf{r}^T}\boldsymbol{\psi} &= \underbrace{\delta\mathbf{u}^T\mathcal{J}'[\mathbf{u}]^T}_{\delta\mathbf{J}}. \\ \delta\mathbf{J} &= \delta\mathbf{r}^T\boldsymbol{\psi}. \end{aligned} \tag{3.6}$$

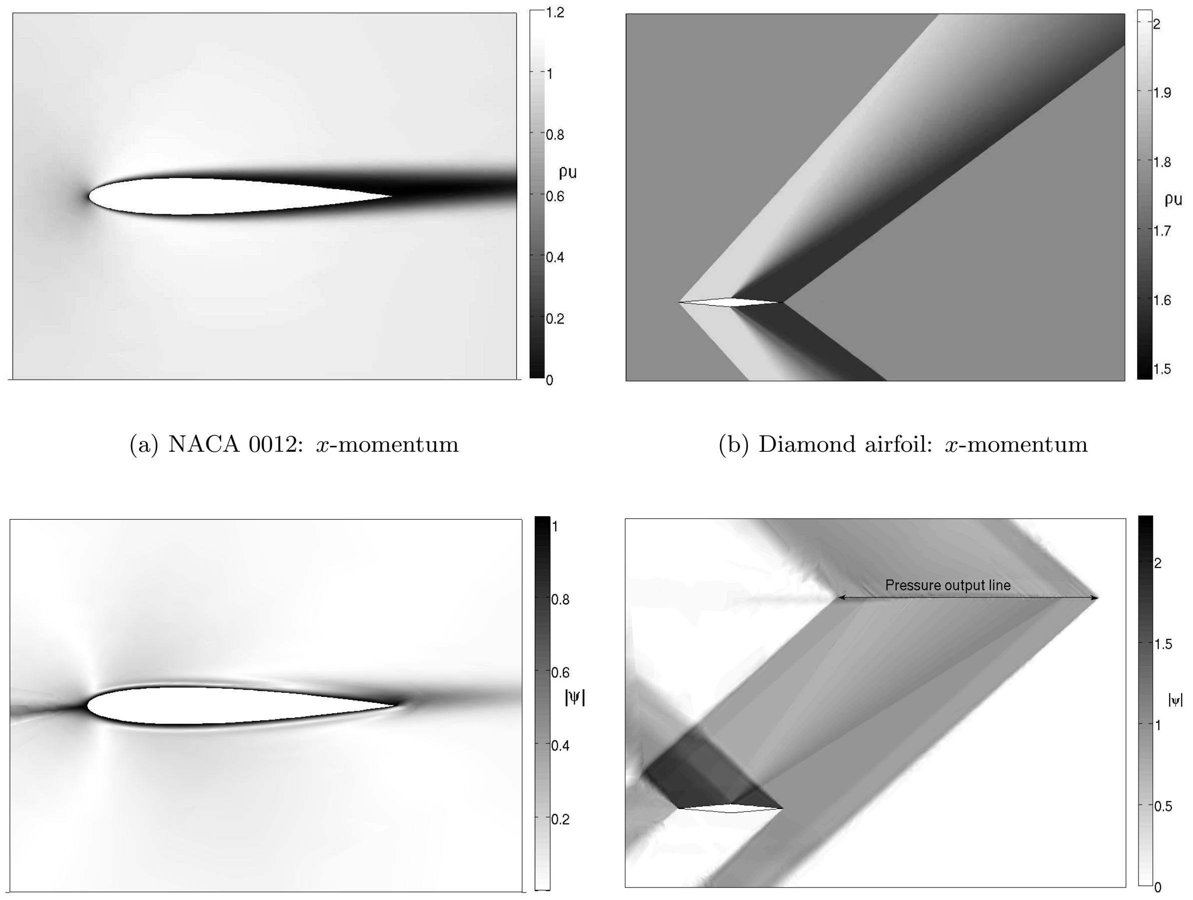
Thus, $\boldsymbol{\psi}$ relates the local residual to the output error. For problems governed by PDEs, (3.6) becomes an integral over the domain, Ω, with $\boldsymbol{\psi}$ acting as a Green's function, e.g. $\delta \mathbf{J} = \int_{\Omega} \delta \mathbf{r}\, \boldsymbol{\psi} d\mathbf{x}$.

Figure 3-1 illustrates sample flow and adjoint solutions. Figure 3-1a shows the x-momentum for a Navier-Stokes solution around a NACA 0012 airfoil at $M = 0.5$, $Re = 5000$, $\alpha = 2^o$. Directly below, in Figure 3-1c, is the associated adjoint solution for a drag output. Note, for K equations (i.e. mass, momentum, energy), the adjoint solution has K components, each relating one of the K residual components to the output error. According to the discussion in the previous paragraph, the output error is affected by residuals in locations where the adjoint is nonzero. As shown in Figure 3-1c, the x-momentum adjoint is nonzero in the boundary layer, in the wake, and in the flow in front of the airfoil leading edge. Therefore, x-momentum residuals in these areas will likely contribute more to the drag error in comparison to x-momentum residuals in other parts of the domain. Figure 3-1b shows another flow solution: the x-momentum for an Euler solution around a diamond airfoil in supersonic, $M = 1.5$ flow. The associated adjoint for a pressure line-integral output is given directly below, in Figure 3-1d. Since information can only propagate downstream at certain angles (Mach wave angles) in supersonic flow, the pressure integral is not affected by residuals in areas from which information cannot propagate to the measurement line. This observation is reflected by the fact that the adjoint solution is zero in these areas. To be precise, for a finite element formulation in which the residual is due to the discretization error, application of Galerkin orthogonality shows that the adjoint interpolation error, rather than the adjoint itself, weights the residual in the output error expression.

3.1.2 Error Estimation and Localization

The adjoint solution can be used to estimate the error in the output and to localize the error to individual elements. The following analysis is based on an abundant number of similar techniques available in the literature. The works of Hartmann and Houston [36], Lu [48], Barth and Larson [4], Giles and Suli [31], Giles and Pierce [30], Becker and Rannacher [9], and Venditti and Darmofal [73] are the most relevant to this analysis. These authors present fundamentally similar ideas with differences primarily in the implementation. In particular, the error estimate used in this work is nearly identical to that used by Hartmann and Houston and by Lu.

The starting point for the error estimation and localization is a more formal definition of the adjoint for a general semi-linear form and for a non-infinitesimal perturbation in the solution. Let $\mathcal{R}_H\big(\mathbf{u}_H, \mathbf{v}_H\big) = 0$, $\forall \mathbf{v}_H \in \mathcal{V}_H$ be a general semi-linear weighted residual statement, such as that obtained in equation (2.6) for the compressible Navier-Stokes



(a) NACA 0012: x-momentum

(b) Diamond airfoil: x-momentum

(c) NACA 0012: x-momentum adjoint

(d) Diamond airfoil: x-momentum adjoint

Figure 3-1: Sample flow and adjoint solutions: (a) x-momentum for a NACA 0012 in subsonic, viscous flow; (b) x-momentum for a diamond airfoil in supersonic, $M = 1.5$ flow; (c) x-momentum adjoint associated with a drag output on the NACA 0012; (d) x-momentum adjoint for a pressure line integral output computed several chords away from the diamond airfoil. For the adjoint, the absolute value is plotted with dark areas indicating large magnitudes.

discretization. $\mathcal{V}_H$ is the discrete approximation space and $\mathcal{R}_H : \mathcal{W}_H \times \mathcal{W}_H \to \mathbb{R}$, where $\mathcal{W}_H \equiv \mathcal{V}_H + \mathcal{V}$ is a space that includes functions in $\mathcal{V}_H$ and $\mathcal{V}$. This space is defined because $\mathcal{V}_H$ is not required to be a subspace of $\mathcal{V}$; in particular, this is the case with DG approximation. For the exact solution, $\mathbf{u}$, the assumption is made that $\mathcal{R}_H\big(\mathbf{u}, \mathbf{w}\big) = 0$, $\forall \mathbf{w} \in \mathcal{W}_H$.

For a nonlinear output, $\mathcal{J}(\mathbf{u})$, the adjoint, or dual, problem reads: find $\boldsymbol{\psi} \in \mathcal{V}$ such that

$$\bar{\mathcal{R}}_H(\mathbf{u}, \mathbf{u}_H; \mathbf{v}, \boldsymbol{\psi}) = \bar{\mathcal{J}}(\mathbf{u}, \mathbf{u}_H; \mathbf{v}), \quad \forall \mathbf{v} \in \mathcal{W}_H. \tag{3.7}$$

The mean value linearizations $\bar{\mathcal{R}}_H : \mathcal{W}_H \times \mathcal{W}_H \to \mathbb{R}$ and $\bar{\mathcal{J}} : \mathcal{W}_H \to \mathbb{R}$ are given by

$$\begin{aligned}\bar{\mathcal{R}}_H(\mathbf{u}, \mathbf{u}_H; \mathbf{v}, \mathbf{w}) &= \int_0^1 \mathcal{R}'_H[\theta\mathbf{u} + (1-\theta)\mathbf{u}_H](\mathbf{v}, \mathbf{w})d\theta, \\ \bar{\mathcal{J}}(\mathbf{u}, \mathbf{u}_H; \mathbf{v}) &= \int_0^1 \mathcal{J}'[\theta\mathbf{u} + (1-\theta)\mathbf{u}_H](\mathbf{v})d\theta,\end{aligned}$$

where $\mathbf{v}, \mathbf{w} \in \mathcal{W}_H$, and the primed notation denotes the Frechét derivative. The semi-linear form and output are assumed to be sufficiently regular such that the adjoint solution exists and is unique. The use of the mean-value linearizations to define the adjoint problem is the typical starting point for non-linear problems [36, 48, 4, 9]. For $\mathbf{v} = \mathbf{u} - \mathbf{u}_H$,

$$\begin{aligned}\bar{\mathcal{R}}_H(\mathbf{u}, \mathbf{u}_H; \mathbf{u} - \mathbf{u}_H, \mathbf{w}) &= \mathcal{R}_H(\mathbf{u}, \mathbf{w}) - \mathcal{R}_H(\mathbf{u}_H, \mathbf{w}), \\ \bar{\mathcal{J}}(\mathbf{u}, \mathbf{u}_H; \mathbf{u} - \mathbf{u}_H) &= \mathcal{J}(\mathbf{u}) - \mathcal{J}(\mathbf{u}_H).\end{aligned}$$

Using $\mathcal{R}_H(\mathbf{u}, \mathbf{w}) = 0$, $\forall \mathbf{w} \in \mathcal{W}_H$, the output error can be expressed as

$$\begin{aligned}\mathcal{J}(\mathbf{u}) - \mathcal{J}(\mathbf{u}_H) &= \bar{\mathcal{J}}(\mathbf{u}, \mathbf{u}_H; \mathbf{u} - \mathbf{u}_H) \\ &= \bar{\mathcal{R}}_H(\mathbf{u}, \mathbf{u}_H; \mathbf{u} - \mathbf{u}_H, \boldsymbol{\psi}) \\ &= 0 - \mathcal{R}_H(\mathbf{u}_H, \boldsymbol{\psi}) \\ &= -\mathcal{R}_H(\mathbf{u}_H, \boldsymbol{\psi} - \boldsymbol{\psi}_H), \end{aligned} \tag{3.8}$$

where $\boldsymbol{\psi}_H \in \mathcal{V}_H$ can be arbitrary at this point due to the Galerkin orthogonality property: $\mathcal{R}_H(\mathbf{u}_H, \mathbf{v}_H) = 0$ for any $\mathbf{v}_H \in \mathcal{V}_H$. Defining an adjoint residual,

$$\bar{\mathcal{R}}^{\psi}_H(\mathbf{u}, \mathbf{u}_H; \mathbf{v}, \mathbf{w}) \equiv \bar{\mathcal{R}}_H(\mathbf{u}, \mathbf{u}_H; \mathbf{v}, \mathbf{w}) - \bar{\mathcal{J}}(\mathbf{u}, \mathbf{u}_H; \mathbf{v}), \qquad \mathbf{v}, \mathbf{w} \in \mathcal{W}_H,$$

the output error can also be expressed as

$$\begin{aligned}\mathcal{J}(\mathbf{u}) - \mathcal{J}(\mathbf{u}_H) &= \bar{\mathcal{R}}_H(\mathbf{u}, \mathbf{u}_H; \mathbf{u} - \mathbf{u}_H, \boldsymbol{\psi}_H) - \bar{\mathcal{R}}^{\psi}_H(\mathbf{u}, \mathbf{u}_H; \mathbf{u} - \mathbf{u}_H, \boldsymbol{\psi}_H) \\ &= \mathcal{R}_H(\mathbf{u}, \boldsymbol{\psi}_H) - \mathcal{R}_H(\mathbf{u}_H, \boldsymbol{\psi}_H) - \bar{\mathcal{R}}^{\psi}_H(\mathbf{u}, \mathbf{u}_H; \mathbf{u} - \mathbf{u}_H, \boldsymbol{\psi}_H) \\ &= -\bar{\mathcal{R}}^{\psi}_H(\mathbf{u}, \mathbf{u}_H; \mathbf{u} - \mathbf{u}_H, \boldsymbol{\psi}_H). \end{aligned} \tag{3.9}$$

As $\mathbf{u}$ and $\boldsymbol{\psi}$ are in general not known, two approximations are employed to make the above output error estimates practical. First, the exact mean-value linearizations are replaced by approximate linearizations about $\mathbf{u}_H$. This approximation is used by Hartmann and Houston and by Lu. Barth and Larson [4] additionally employ a more accurate approximation to the mean-value linearization using numerical integration between $\mathbf{u}_H$ and a reconstructed approximation to $\mathbf{u}$. However, the results do not indicate a substantial improvement in the accuracy of the error estimate. To minimize errors in (3.8) and (3.9) due to no longer using mean-value linearizations, $\boldsymbol{\psi}_H$ is set to the finite element approximation of $\boldsymbol{\psi}$ [31]. That is, $\boldsymbol{\psi}_H$ satisfies

$$\mathcal{R}_H^{\psi}(\mathbf{u}_H; \mathbf{v}_H, \boldsymbol{\psi}_H) = 0, \quad \forall \mathbf{v}_H \in \mathcal{V}_H, \tag{3.10}$$

where $\mathcal{R}_H^{\psi}(\mathbf{u}_H; \mathbf{v}, \mathbf{w})$ is the adjoint residual computed with linearization only about $\mathbf{u}_H$:

$$\mathcal{R}_H^{\psi}(\mathbf{u}_H; \mathbf{v}, \mathbf{w}) = \mathcal{R}'_H[\mathbf{u}_H](\mathbf{v}, \mathbf{w}) - \mathcal{J}'[\mathbf{u}_H](\mathbf{v}), \qquad \mathbf{v}, \mathbf{w} \in \mathcal{W}_H.$$

In the vicinity of under-resolved flow features or shocks, the error due to this approximate linearization is not guaranteed to be small. Care must be taken in working with error estimates obtained in these cases.

The second approximation consists of replacing the exact solution errors $\mathbf{u} - \mathbf{u}_H$ and $\boldsymbol{\psi} - \boldsymbol{\psi}_H$ by $\mathbf{u}_h - \mathbf{u}_H$ and $\boldsymbol{\psi}_h - \boldsymbol{\psi}_H$, respectively, where $\mathbf{u}_h$ and $\boldsymbol{\psi}_h$ are approximations to $\mathbf{u}$ and $\boldsymbol{\psi}$ on an enriched finite element space, $\mathcal{V}_h$. A common approach for choosing $\mathcal{V}_h$ consists of adding degrees of freedom to $\mathcal{V}_H$ by uniformly refining the mesh and/or by increasing the interpolation order, p [8, 30, 73, 48]. The motivation for using a richer space for $\mathcal{V}_h$ is to increase the accuracy of $\mathbf{u}_h$ and $\boldsymbol{\psi}_h$ and, hence, to improve the error estimate. However, the richer the space $\mathcal{V}_h$, the more costly the estimator, with uniform grid refinement becoming particularly expensive and cumbersome for practical three-dimensional simulations. For example, in three dimensions, a quasi-uniform tetrahedral refinement leads to a five-fold increase in the degrees of freedom, whereas an order increase from $p = 2$ to $p = 3$ leads to a two-fold increase. Thus, in this work, $\mathcal{V}_h$ is constructed from $\mathcal{V}_H$ by increasing the order to $p+1$ while keeping the mesh fixed. The primal and adjoint solutions are then reconstructed on $\mathcal{V}_h$ in a patch-wise manner. Lu [48], Barth and Larson [4], and Rannacher [64] employ this approach and obtain satisfactory results for numerous cases. Hartmann and Houston [36] also use an order $p + 1$ space for the dual solution, but rather than reconstructing the dual, they solve for it on the enriched space. Solín and Demkowicz [68] extend this idea by solving both the primal and dual problems on a uniformly-refined mesh, $h = H/2$, at order $p + 1$. While these approximations are more accurate, they are also more expensive.

The approximations $\mathbf{u}_h$ and $\boldsymbol{\psi}_h$ are created by a reconstruction process on $\mathcal{V}_h$. Local H_1 patch reconstruction is used, in which the minimized error for each element $\kappa \in T_H$ takes the form

$$E_\kappa^2(\mathbf{v}_\kappa, \mathbf{u}_H) = \sum_{l \in \mathcal{P}_\kappa} \Big(\int_l (\mathbf{v}_\kappa - \mathbf{u}_H)^2 d\mathbf{x} + \sum_{i=1}^{d} c_i \int_l (\partial_i \mathbf{v}_\kappa - \partial_i \mathbf{u}_H)^2 d\mathbf{x} \Big),$$

where $\mathcal{P}_\kappa$ is the patch of neighboring elements in T_H (including κ), $\mathbf{v}_\kappa \in P^{p+1}(\mathcal{P}_\kappa)$ denotes the order $p+1$ reconstructed solution on the patch, d is the dimension, and the c_i are $O(\Delta x_i^2)$ scaling coefficients specific to each element, determined by the dimensions of the elemental bounding boxes. An example of a patch of elements, $\mathcal{P}_\kappa$, associated with element κ is illustrated for two dimensions in Figure 3-2. Effectively, the reconstruction process creates a smooth representation of the solution, minimizing in a least squares sense the error in value and slope relative to the existing solution, $\mathbf{u}_H$, on the patch. In practice, the least-squares error minimization is performed via a QR factorization. The reconstructed solution, $\mathbf{u}_h$, is set according to $\mathbf{u}_h|_\kappa = \mathbf{v}_\kappa|_\kappa$; that is, the reconstruction is performed separately for each element. $\boldsymbol{\psi}_h$ is obtained analogously. To further improve the approximation, one element-Jacobi smoothing iteration is performed on $\mathbf{u}_h$ and $\boldsymbol{\psi}_h$.

Using $\mathbf{u}_h$ and $\boldsymbol{\psi}_h$ in place of $\mathbf{u}$ and $\boldsymbol{\psi}$ in (3.8) and (3.9) yields the following approximations to the output error (making use of $\mathcal{V}_H \subset \mathcal{V}_h$):

$$\begin{aligned} \mathcal{J}(\mathbf{u}) - \mathcal{J}(\mathbf{u}_H) &\approx -\mathcal{R}_h(\mathbf{u}_H, \boldsymbol{\psi}_h - \boldsymbol{\psi}_H) = -\sum_{\kappa \in T_H} \mathcal{R}_h(\mathbf{u}_H, (\boldsymbol{\psi}_h - \boldsymbol{\psi}_H)|_\kappa), \\ \mathcal{J}(\mathbf{u}) - \mathcal{J}(\mathbf{u}_H) &\approx -\mathcal{R}_h^\psi(\mathbf{u}_H; \mathbf{u}_h - \mathbf{u}_H, \boldsymbol{\psi}_H) = -\sum_{\kappa \in T_H} \mathcal{R}_h^\psi(\mathbf{u}_H; (\mathbf{u}_h - \mathbf{u}_H)|_\kappa, \boldsymbol{\psi}_H). \end{aligned}$$

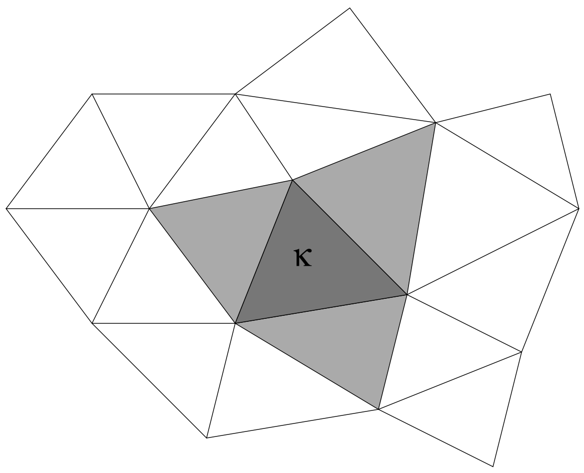


Figure 3-2: Patch of elements, $\mathcal{P}_\kappa$, associated with element κ. $\mathcal{P}_\kappa$ consists of κ and the adjacent elements (shaded). The reconstructed primal and adjoint solutions on κ are based on an H_1 error minimization over $\mathcal{P}_\kappa$.

In the above expressions, $|_\kappa$ refers to restriction to element κ, and T_H is the triangulation. A local error indicator on each element is obtained by averaging the local primal-residual and adjoint-residual contributions to the output error in the above expressions. Specifically, in this work, the error indicator in each element κ is taken to be

$$\epsilon_\kappa = \frac{1}{2}\Big(\big|\mathcal{R}_h(\mathbf{u}_H, (\boldsymbol{\psi}_h - \boldsymbol{\psi}_H)|_\kappa)\big| + \big|\mathcal{R}_h^\psi(\mathbf{u}_H; (\mathbf{u}_h - \mathbf{u}_H)|_\kappa, \boldsymbol{\psi}_H)\big|\Big). \tag{3.11}$$

For systems of equations, indicators are computed separately for each equation and summed together. The global output error estimate, $\epsilon = \sum_\kappa \epsilon_\kappa$, is not a bound on the actual error in the output because of the approximations made in the derivation. However, the validity of the approximations is expected to increase as $\mathbf{u}_H \to \mathbf{u}$. In the literature, various other indicators are presented, using either/both the primal-based and the dual-based error estimate expressions [73, 9, 36, 31, 4]. Using a combination of both expressions targets errors in both the primal and the dual solutions and has been found sufficiently effective in driving adaptation.

3.2 Adaptation Strategy

Given a localized error estimate, an adaptive method modifies the computational mesh in an attempt to decrease and equidistribute the error. As mentioned in Section 1.2.2, in high-order finite element methods, possible adaptation strategies include p, h, and hp, where p-adaptation refers to changing only the order of interpolation, h-adaptation refers to changing only the computational mesh, and hp-adaptation is a combination of both.

The adaptation strategy chosen for this work is h-adaptation at a constant p. h-adaptation was chosen over pure p-adaptation to allow for efficient resolution of singularities and areas of anisotropy and to avoid the requirement of a reasonable starting mesh. Compared to hp-adaptation, this strategy does not take advantage of the cost savings associated with optimally choosing h versus p adaptation. However, it avoids the regularity decision, which requires either a solution regularity estimate or a heuristic algorithm. Extending the proposed methods to hp-adaptation is one of the possible areas of future work.

The h-adaptation method consists of high-order anisotropy detection and mesh optimization. Inputs to this method are the current mesh, the solution on the mesh, an element-local error indicator, and a user-specified error tolerance. The output is a mesh-sizing request in the form of a metric associated with each element for use in re-meshing. The following two sections describe the high-order anisotropy detection and mesh optimization strategies.

3.2.1 Anisotropy in High-Order Solutions

An important ingredient in h-adaptation for aerodynamic computations is the ability to generate stretched elements in areas where the solution exhibits anisotropy, which is variation of disparate magnitudes in different directions. For $p = 1$, the dominant method for detecting anisotropy involves estimating the Hessian matrix, $\mathbf{H}$, of a scalar solution, u [60, 32, 17]. The components of $\mathbf{H}$ are given by

$$H_{ij} = \frac{\partial^2 u}{\partial x_i \partial x_j}, \quad i, j \in [1, .., d].$$

The second derivatives can be estimated by, for example, a quadratic reconstruction of the linear solution. For the Euler or Navier-Stokes equations, the Hessian of the Mach number has been found to perform sufficiently well. Of course, other scalar quantities may also be suitable, and perhaps the most effective choice is an average or minimum Hessian of several scalar quantities [17]. In this work, use of the Mach number for u has produced acceptable results.

The Hessian matrix is used to define a Riemannian metric, the idea being that in an optimal mesh, all edge lengths will have unit measure under the metric [17, 32]. In a Cartesian coordinate system, the length of an infinitesimal segment $\delta\mathbf{x}$ under a Riemannian metric $\mathbf{M}$ is given by

$$\delta\Gamma = \delta\mathbf{x}^T \; \mathbf{M} \; \delta\mathbf{x} = \delta x_i \; M_{ij} \; \delta x_j, \tag{3.12}$$

where δx_i are the components of the length-d vector $\delta\mathbf{x}$ and M_{ij} are the components of the $d \times d$ matrix $\mathbf{M}$. The metric is obtained from the Hessian by requiring that the interpolation error estimate of the scalar quantity u is the same in any chosen spatial direction. For linear interpolation of a scalar quantity along the segment $\delta\mathbf{x}$, the maximum interpolation error can be bounded by (see for example [69])

$$\max_{\mathbf{x} \in \delta\mathbf{x}} |u(\mathbf{x}) - u_H(\mathbf{x})| \leq C_1 |\delta\mathbf{x}|^2 \max_{\mathbf{x} \in \delta\mathbf{x}} \left| u^{(2)}_{\delta\mathbf{x}}(\mathbf{x}) \right|, \tag{3.13}$$

where $u_H(\mathbf{x})$ is the linear interpolant of u, C_1 is a constant independent of u, and $u^{(2)}_{\delta\mathbf{x}}$ is the second derivative of u evaluated in the direction of $\delta\mathbf{x}$. Requiring the interpolation error bound to be approximately constant, independent of the direction of $\delta\mathbf{x}$, yields

$$|\delta\mathbf{x}|^2 \left| u^{(2)}_{\delta\mathbf{x}} \right| = C_2 = \text{const.}$$

The left-hand side can be bounded in terms of the Hessian,

$$|\delta\mathbf{x}|^2 u^{(2)}_{\delta\mathbf{x}} = \delta\mathbf{x}^T\mathbf{H}\delta\mathbf{x} \quad\Rightarrow\quad |\delta\mathbf{x}|^2 \left|u^{(2)}_{\delta\mathbf{x}}\right| \leq \delta\mathbf{x}^T|\mathbf{H}|\delta\mathbf{x}, \tag{3.14}$$

where $|\mathbf{H}|$ is the positive, semi-definite form of the Hessian: $|\mathbf{H}| = \mathbf{V}|\mathbf{\Lambda}|\mathbf{V}^{-1}$ for $\mathbf{H} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$. This bound on the interpolation error is used to define the Riemannian metric. Comparing (3.14) and (3.12), the requirement that the interpolation error estimate be independent of the direction of $\delta\mathbf{x}$ can be expressed as a requirement of equal measure of $\delta\mathbf{x}$ under the metric

$$\mathbf{M} = C|\mathbf{H}|, \tag{3.15}$$

where C is a constant independent of direction. Two intervals, $\delta\mathbf{x}_1$ and $\delta\mathbf{x}_2$, having the same measure under this $\mathbf{M}$ will have the same estimated interpolation error bounds.

The metric $\mathbf{M}$ contains information on the desired mesh edge lengths in physical space. As $\mathbf{M}$ is symmetric and positive, semi-definite, the unit measure requirement,

$$\mathbf{x}^T\ \mathbf{M}\ \mathbf{x} = 1,$$

describes an ellipsoid in physical space. The eigenvectors of $\mathbf{M}$, $\mathbf{e}_i$, are the orthogonal axes of the ellipsoid – i.e. the principal directions. The corresponding eigenvalues, λ_i, are related to the lengths of the axes, h_i, via

$$\lambda_i = \frac{1}{h_i^2}.$$

Physically, the h_i are the principal stretching magnitudes. In this work, the eigenvalues are ordered from largest to smallest so that λ_1 is the largest eigenvalue, and, hence, h_1 is the smallest stretching magnitude. A diagram of an ellipse resulting from the unit-measure requirement in two dimensions is given in Figure 3-3.

The constant C in (3.15) controls the absolute magnitude of the requested mesh sizes, h_i. In standard Hessian-based adaptation, C is constant over the computational domain, and its value is often set heuristically [17]. Regardless of the value of C, however, the Hessian matrix contains information on the relative mesh sizes,

$$h_i/h_j = (\lambda_j/\lambda_i)^{1/2}.$$

Note that, from (3.15), the ratio of eigenvalues of $\mathbf{M}$ equals the ratio of eigenvalues of $|\mathbf{H}|$.

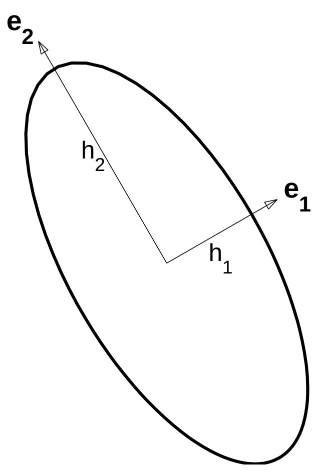


Figure 3-3: Ellipse representing requested mesh sizes implied by equal measure under a Riemannian metric $\mathbf{M}$. Also shown are the principal directions, $\mathbf{e}_i$, and the associated principal stretching magnitudes, h_i.

Unfortunately, anisotropy detection based on the standard Hessian matrix is not suited for $p > 1$ interpolation, because the interpolation error estimate in (3.13) assumes linear interpolation of u. An example demonstrating the problem of applying Hessian-based anisotropy detection to a higher-order solution is given in Appendix B. For a general, order p interpolant, u_H, the equivalent interpolation error estimate reads

$$\max_{\mathbf{x}\in\delta\mathbf{x}} |u(\mathbf{x}) - u_H(\mathbf{x})| \leq C_3 |\delta\mathbf{x}|^{p+1} \max_{\mathbf{x}\in\delta\mathbf{x}} \left| u^{(p+1)}_{\delta\mathbf{x}}(\mathbf{x}) \right| . \tag{3.16}$$

That is, the $p+1$st derivatives of u govern the inability of an order p interpolant to interpolate the exact solution. Thus, the Riemannian metric (i.e. stretching ratios, h_i/h_j, and principal directions, $\mathbf{e}_i$) should be based on estimates of the $p+1$st derivatives. An important assumption implicit in this estimate is that the analytical solution is sufficiently regular. For general compressible flows, this assumption may need to be modified in the presence of discontinuities or singularities.

While second derivatives can be arranged into a symmetric $d \times d$ Hessian matrix, this is no longer the case with $p+1$st derivatives. Thus, (3.15) cannot be used for the metric definition. One solution, used in this work, is to construct a metric by explicitly identifying orthogonal principal stretching directions and associated magnitudes. Specifically, let $\mathbf{e}_1$ be the direction of maximum $p+1$st derivative and $\mathbf{e}_2$ the direction of maximum $p+1$st derivative in the plane orthogonal to $\mathbf{e}_1$. Under this definition, the final direction, $\mathbf{e}_d$, is fully-determined by the previous directions. The $p+1$st derivative in a general direction is calculated by applying a coordinate transformation to $p + 1$st derivatives in a fixed frame. Since an order p interpolant contains no information on $p + 1$st derivatives, these derivatives must be approximated. In this work, the derivatives are calculated on the order $p + 1$ reconstructed solution created for error estimation (Section 3.1). Thus, the $p + 1$st derivatives are constant over each element.

The direction of maximum $p + 1$st derivative is calculated by an exhaustive search combined with bisection. For example, in two dimensions, an angle range is discretized into n_θ intervals, and the $p+1$st derivative is calculated in each discrete direction. The interval around the maximum calculated value serves as the angle range for n_b bisection iterations that hone in on the maximum derivative value. Default values for the search parameters in two dimensions are $n_\theta = 36$ and $n_b = 15$. While this exhaustive search has not been a bottleneck for the test cases in this work, a more efficient option consists of identifying local extrema of the $p + 1$st derivative by differentiating the transformation expression with respect to the transformation parameters (e.g. one angle in 2D or two angles in 3D). This approach requires solving one polynomial equation in 2D and two coupled polynomial equations in 3D. The maximum $p + 1$st derivative can then be found by evaluating the transformation expression in the local extrema directions.

By construction, the $\mathbf{e}_i$ directions are orthogonal and, therefore, suitable for specifying the metric of directional sizes. The stretching magnitudes are obtained by an equal interpolation error estimate from (3.16). Equidistributing the interpolation error in each principal direction yields

$$h_i^{p+1} u_{\mathbf{e}_i}^{(p+1)} = \text{const.} \quad \Rightarrow \quad \frac{h_i}{h_j} = \left(u_{\mathbf{e}_j}^{(p+1)} / u_{\mathbf{e}_i}^{(p+1)} \right)^{1/(p+1)}. \tag{3.17}$$

As with Hessian-based analysis, the result in (3.17) provides only the relative mesh sizing. The absolute magnitudes for h_i are based on the error indicator, as described in the following section. The final metric, $\mathbf{M}$, is fully specified by the d eigenvectors, $\mathbf{e}_i$, and associated eigenvalues, $\lambda_i = 1/h_i^2$.

3.2.2 Mesh Optimization

In h-adaptation, mesh optimization refers to deciding which elements to refine or coarsen and/or the amount of refinement or coarsening. The optimization has important implications for practical simulations: too little refinement at each adaptation iteration may result in an unnecessary number of iterations; too much refinement may ask for an expensive solve on an overly-refined mesh.

Many of the current adaptation strategies rely on some variation of the fixed-fraction method [40, 68, 36], in which a prescribed fraction of elements with the highest error indicator is refined. While adequate for testing and small cases, this method poses an automation and efficiency problem for practical simulations due to the often *ad-hoc* fixed-fraction parameter. More sophisticated optimization strategies attempt to meet the global tolerance while equidistributing the error among elements. Zienkiewicz and Zhu [80] define a permissible element error $e_\kappa = e_0/N$ at each adaptation iteration, where e_0 is the global tolerance, and N is the current number of elements. Coupled with an *a priori* error estimate, this "refinement prediction" method yields element sizing at each adaptation iteration. Venditti and Darmofal [73, 74], employ a similar approach and extend it to anisotropic sizing using the Hessian matrix. Compared to the fixed-fraction method, refinement prediction has the advantage that it specifies the magnitude of refinement in each element.

For elliptic problems, Rannacher *et al* [64, 9] present another mesh optimization strategy in which an optimal mesh size function is constructed continuously over the entire domain. The construction is based on solving a constrained minimization problem with a Lagrangian method. Details can be found in the references and in an earlier work by Brandt [13]. Key to this method is an assumption regarding the existence of a mesh-independent function in an expression for the global error. The authors note that this is a heuristic assumption, and that the existence of this function can be rigorously justified only under very restrictive conditions [9].

Anisotropy detection introduces another variable into the mesh optimization process; namely, the stretching of the elements. As mentioned in Section 3.2.1, in "pure" Hessian-based adaptation, the absolute magnitude of stretching is controlled by an arbitrary global scaling factor: the constant C in (3.15). Venditti and Darmofal use an output-based error indicator to determine the length magnitude and obtain a more robust adaptation process [74]. Formaggia *et al* [26] have combined Hessian-based interpolation error estimates with output-based *a posteriori* error analysis to arrive at output-based anisotropic error estimates.

This work adopts a variation of the refinement prediction method of Zienkiewicz and Zhu, modified to allow for mesh anisotropy. One drawback of straightforward refinement prediction is the fact that error equidistribution is performed over the current mesh as opposed to some reasonable prediction of the adapted mesh. While in the asymptotic limit the current and the predicted mesh will converge, by attempting to equidistribute the error on the predicted mesh, adaptive convergence can be accelerated. An example of this effect is demonstrated in Appendix C.

Equidistributing the error on the adapted mesh involves a prediction of the number of elements, N_f, in the adapted (fine) mesh. Let n_κ be the number of fine-mesh elements contained in element κ. n_κ need not be an integer, and $n_\kappa < 1$ indicates coarsening. Let h_i^c denote the current element sizes of κ in principal directions $\mathbf{e}_i^c$, where again i indexes the spatial dimension. If h_i are the requested element sizes in principal directions $\mathbf{e}_i$, n_κ can be approximated as

$$n_\kappa = \prod_i \left(h_i^c / h_i\right). \tag{3.18}$$

The current sizes, h_i^c, are calculated as the singular values of the mapping from a unit equilateral triangle/tetrahedron to element κ. In practice, this mapping is calculated using reference-element Jacobian mappings. If $\mathbf{J}_\kappa$ is the mapping between the reference element and κ, and $\mathbf{J}_E$ is the mapping between the reference element and the unit equilateral triangle/tetrahedron, then $\mathbf{J}_\kappa \mathbf{J}_E^{-1}$ is the mapping of interest. This mapping is illustrated for two dimensions in Figure 3-4. The singular values of $\mathbf{J}_\kappa \mathbf{J}_E^{-1}$ are the stretching magnitudes in each of the principal directions. The resulting grid-implied metric is similar to that used

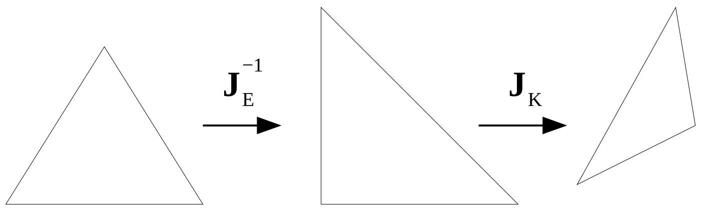


Figure 3-4: Mapping from unit equilateral triangle to a general triangle, κ, via reference, right-triangle Jacobians. Singular values of this mapping serve as the principal stretching directions.

by Venditti [72]. Such a calculation ensures that an isotropic metric is retained for a mesh of equilateral triangles. Note, (3.18) is based on an approximate volume comparison between the current and refined elements and, therefore, does not depend on the principal directions $\mathbf{e}_i^c$ or $\mathbf{e}_i$

To satisfy error equidistribution, each fine-mesh element is allowed an error of e_0/N_f, which means that each element κ is allowed an error of $n_\kappa e_0/N_f$. By relating changes in element size to expected changes in the local error, an expression for n_κ is obtained, from which the absolute element sizes, h_i, follow. In this work, an *a priori* estimate for the output error serves as this relation,

$$\frac{\epsilon_\kappa}{\epsilon_\kappa^c} = \left(\frac{h_1}{h_1^c}\right)^{r_\kappa}, \tag{3.19}$$

where ϵ_κ^c is the current error indicator, ϵ_κ is the expected error indicator following adaptation, and r_κ is the expected convergence rate of the error indicator. Formally, $r_\kappa = s+t-2$ for elliptic problems, and $r_\kappa = s+t-1$ for first-order hyperbolic problems, where $1 \le s \le \min(p+1, \gamma_\kappa)$ is the convergence of the primal solution, and $1 \le t \le \min(p+1, \gamma_\kappa^\Psi)$ is the convergence of the dual solution [34]. γ_κ and γ_κ^Ψ are the regularities of the primal and dual solutions, respectively. In this work, r_κ is set using the asymptotic estimates, $s = \min(p+1, \gamma_\kappa)$ and $t = \min(p+1, \gamma_\kappa^\Psi)$, and the regularities are assumed to be at least $p+1$ on elements away from geometric singularities (i.e. corners). On elements adjacent to or containing geometric corners (e.g. a trailing edge of an airfoil), numerical experiments indicate that r_κ is at most 1 for both hyperbolic and elliptic problems. Thus, geometric corners are detected for both boundary-conforming and cut-cell meshes and r_κ is limited to 1 on adjacent elements.

The *a priori* estimate in (3.19) is asymptotically valid for many common engineering outputs, including forces and pressure distribution norms. In the estimate, the error is assumed to scale with h_1, which corresponds to the direction of maximum $p+1$st derivative. Implicit in the estimate is that the principal directions corresponding to the requested size, h_1, and the current size, h_1^c, align. One option for accounting for a difference in principal directions is to replace h_1^c in (3.19) with $h^c(\mathbf{e}_1)$, the current, grid-implied size in the requested principal direction $\mathbf{e}_1$. However, as $h_1^c \le h^c(\mathbf{e})$ for any direction, $\mathbf{e}$, using h_1^c in (3.19) leads to a more conservative estimate for h_1 in the early stages of adaptation. Furthermore, the assumption that h_1 and h_1^c align becomes more valid as the adaptation progresses. Equating the allowable error with the expected error from the *a priori* estimate yields

$$\underbrace{n_\kappa \frac{e_0}{N_f}}_{\text{allowable error}} = \underbrace{\epsilon_\kappa^c \left(\frac{h_1}{h_1^c}\right)^{r_\kappa}}_{\textit{a priori} \text{ estimate}} . \tag{3.20}$$

Expressing $\frac{h_1}{h_1^c}$ in terms of n_κ and the known relative sizes (3.17) yields a relation between n_κ and N_f. For example, in two dimensions,

$$n_\kappa = \frac{h_1^c}{h_1}\frac{h_2^c}{h_2} \quad \Rightarrow \quad \frac{h_1}{h_1^c} = \left(\frac{1}{n_\kappa}\frac{h_1}{h_2}\frac{h_2^c}{h_1^c}\right)^{1/2},$$

where both h_1/h_2 and h_2^c/h_1^c are known. Substituting for h_1/h_1^c into (3.20),

$$n_\kappa \frac{e_0}{N_f} = \epsilon_\kappa^c \left(\frac{1}{n_\kappa}\frac{h_1}{h_2}\frac{h_2^c}{h_1^c}\right)^{r_\kappa/2} \quad \Rightarrow \quad n_\kappa^{1+r_\kappa/2} = \frac{\epsilon_\kappa^c}{e_0/N_f}\left(\frac{h_1}{h_2}\frac{h_2^c}{h_1^c}\right)^{r_\kappa/2}.$$

Using $N_f = \sum_\kappa n_\kappa$,

$$N_f = \sum_\kappa n_\kappa = \sum_\kappa \left[\left(\frac{\epsilon^c_\kappa}{e_0/N_f} \right)^{\frac{2}{r_\kappa+2}} \left(\frac{h_1}{h_2} \frac{h^c_2}{h^c_1} \right)^{\frac{r_\kappa}{r_\kappa+2}} \right]. \tag{3.21}$$

A similar expression is obtained in three dimensions. If the r_κ are the same for every element, this equation can be solved directly for N_f. Otherwise, it is solved iteratively. With N_f known, (3.20) yields n_κ, from which the h_i are calculated using (3.18) and (3.17). Thus, the output error estimate in conjunction with the mesh optimization step provides the remaining piece of information necessary to fully specify the directional sizes, h_i. These sizes, along with the principal directions $\mathbf{e}_i$, yield the Riemannian metric for every element.

Mesh optimization is performed at every adaptation iteration following output-error estimation. The adaptation iterations stop when the total error estimate drops below the requested tolerance: $\epsilon \equiv \sum_\kappa \epsilon_\kappa \le e_0$. In practice, two parameters are used to control the behavior of the optimization and adaptation algorithm: the target error fraction, $0 < \eta_t \le 1$, and the adaptation aggressiveness, $0 \le \eta_a < 1$. Specifically, instead of e_0 in (3.20), a modified requested error level, $\tilde{e}_0$ is used, where

$$\tilde{e}_0 = \max\left(\eta_a \epsilon, \eta_t e_0\right).$$

η_t prevents the adaptation convergence from stalling as the error estimate, ϵ, approaches the tolerance, e_0. The aggressiveness parameter, η_a, controls how quickly the error is reduced when the error estimate is far from e_0, and hence may not be very accurate. A value close to zero indicates aggressive adaptation, which has the danger of over-refinement, while a value close to 1 may require an excessive number of adaptation iterations to converge. Default values for these parameters that have been found to work well over a variety of cases are $\eta_t = 0.7$ and $\eta_a = 0.25$.

3.2.3 Implementation

The adaptive solution process is illustrated in Figure 3-5. The input is an initial coarse mesh along with an error tolerance for an output. The iterative process starts by solving the primal and adjoint problems on the initial coarse mesh. Next, the output error is estimated and localized to the elements. If the global error tolerance criterion is met, the adaptive process terminates. Otherwise, the local error indicators are converted to mesh size requests using anisotropy detection and mesh optimization. The computational domain is re-meshed using the requested metric, and the solution on the new mesh is initialized by a transfer of the solution from the old mesh. The process then repeats.

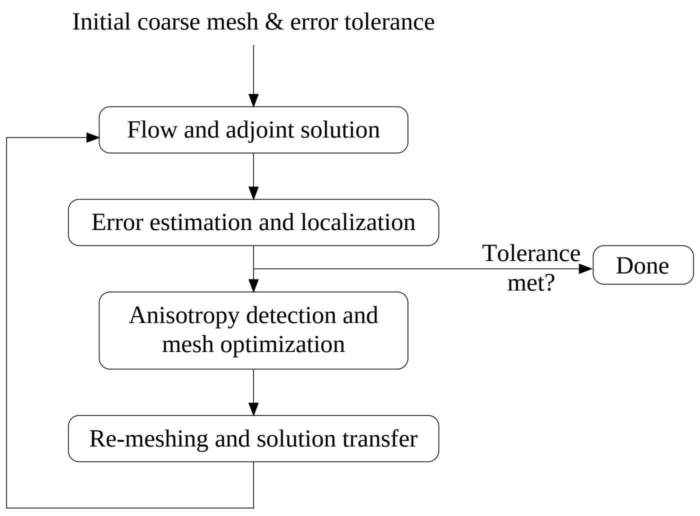

Initial coarse mesh & error tolerance
Flow and adjoint solution
Error estimation and localization
Tolerance met?
Done
Anisotropy detection and mesh optimization
Re-meshing and solution transfer

Figure 3-5: Adaptive solution process flowchart. The input consists of an initial coarse mesh and a requested error tolerance. Adaptation stops when the error tolerance is met. In practice, the most expensive step is the flow and adjoint solution on each mesh.

The primal problem is solved using a preconditioned Newton-GMRES (Generalized Minimal Residual) method, where one of several preconditioners is used in practice. These preconditioners include a line smoother and an incomplete LU smoother, used alone or combined with linear p-multigrid [23]. The default preconditioner used in this work is linear p-multigrid with line smoothing. The adjoint problem is solved sequentially after the primal solve, costing one extra linear solve (or more for multiple outputs). Preconditioned GMRES is used for the adjoint linear solve. Adjoint capability for an output requires only the specification of that output and its linearization, $\mathcal{J}'[\mathbf{u}_H](\mathbf{v})$, as the transpose of the discrete primal Jacobian matrix is used for the adjoint solve.

In two dimensions, meshing of the domain is performed via the Bi-dimensional Anisotropic Mesh Generator (BAMG) [12], which takes as input a mesh with the requested metric defined at the input mesh nodes and produces a new mesh based on the requested metric. The BAMG input mesh is constructed by first uniformly refining the mesh twice and assigning to each subelement the same metric as the original element. Next, on the refined mesh, the metric components at each node are set to the arithmetic mean of the metric components of the adjacent elements since BAMG requires the metric prescribed at the nodes. The uniform refinement of the mesh is necessary to prevent smoothing-out of small mesh size requests in the element-to-node metric transfer. In three dimensions, meshing is performed using TetGen [67], which currently supports only isotropic mesh refinement. Mesh optimization for isotropic refinement is performed as described in this chapter, using one mesh size parameter, $h = h_i$. TetGen creates adapted meshes based on volume requests for each element. These volume requests follow directly from the current, h^c, and requested, h, mesh sizes.

To improve robustness and to speed up convergence, the solution on the new mesh is initialized via a transfer of the solution from the previous mesh. The transfer is performed using an L_2 projection of the state. Specifically, the error minimized in the transfer takes the form

$$E^2 = \int_{\kappa'} (u' - u)^2 d\mathbf{x}, \tag{3.22}$$

where κ' is an element on the new mesh, u' is the solution on κ', and u is the solution on the current mesh. E^2 is minimized using QR factorization, with the state representation coefficients on κ' as the unknowns.

For solutions with large inter-element jumps (e.g. on coarse meshes), such a projection may produce a non-physical state on the new mesh. In such cases, detected by testing for non-physical states at quadrature points, a $p = 0$ restriction of the solution from the previous mesh is performed. This $p = 0$ restriction is always performed on cut cells, which are described in the next section.

Chapter 4

Cut Cells in Two Dimensions

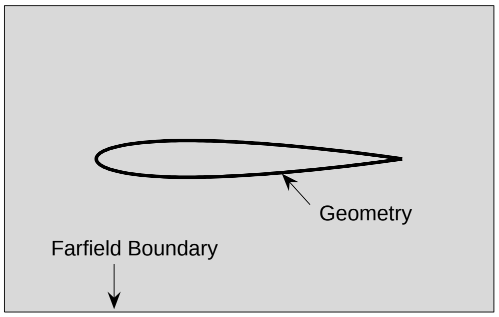
4.1 Cutting and Integration Mechanics

One goal of this thesis is to explore the feasibility of using simplex, cut-cell meshes in a discontinuous Galerkin finite element framework. As mentioned previously in Section 1.1, the motivation for simplex cut cells is to improve meshing robustness, to automate mesh generation for complex geometries, and to allow for general anisotropic meshes. For high-order DG approximations, a cut-cell method relies on robust intersection of triangles with curved geometries and on accurate integration on the resulting cut cells. The following sections describe one implementation of such a cut-cell method in two dimensions.

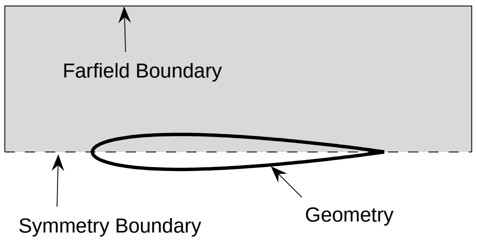
4.1.1 Geometry Definition and Initial Mesh

In two dimensions, a relatively simple but effective geometry representation consists of cubic splines. A cubic spline is an interpolated fit of an ordered set of points, or knots, with the property that the first and second derivatives (i.e. slope and curvature) are continuous across the knots. On spline segments between adjacent knots, the spline coordinates are analytical cubic functions of one parameter, which is usually taken to be the arc length along the spline. Geometric corners, where the tangent vector is discontinuous, can be represented using multiple splines. At spline endpoints, boundary conditions such as zero second or third derivative can be imposed in each coordinate.

Spline geometry representation was chosen in this work because geometry interrogation and intersection with line segments can be performed analytically. The orientation of the splines determines the interior versus exterior of the computational domain; in particular, the computational domain is always on the left as the splines are traversed in the direction of increasing spline parameter value. The computational domain is also bounded by a set of farfield or symmetry boundaries. A common farfield boundary is a rectangular box around the embedded object (Figure 4-1a), although cuts through symmetry boundaries are also allowed (Figure 4-1b). The area enclosed by these farfield/symmetry boundaries is referred to as the “background domain.” It includes the computational domain, where the solution is defined, as well as the interior of the embedded geometry, where no solution exists. An initial mesh of the background domain consists of a coarse triangulation without regard to the embedded objects. If desired, subsequent geometry-adapted triangulations are constructed by refining elements that intersect the splines. Currently, this refinement consists of isotropically decreasing the mesh size request by a prescribed factor and re-meshing the domain a prescribed number of times or until a maximum number of elements is reached. The details of geometry adaptation are not crucial, as only a reasonable starting mesh is sought for the solution-adaptive method.



(a) Single farfield boundary



(b) Farfield and symmetry boundaries

Figure 4-1: Sample farfield and symmetry boundary placements for a cut-cell airfoil case. The background domains are denoted by the shaded areas. As shown in (b), the geometry may cut through the boundary of the background domain, in which case the geometry lying outside the background domain is discarded.

4.1.2 Cutting Algorithm

Given an area-filling mesh of the background domain and a set of splines defining the geometry, a cutting algorithm is employed to determine which elements are cut by the splines and the precise geometry of the cuts. The cutting algorithm proceeds by solving cubic intersection problems (described in Appendix D) to determine intersections of spline segments with element edges and nodes. Careful attention must be given to conditioning for node and tangency intersections. The intersections are performed once and stored for the entire mesh to prevent floating point errors in repeating calculations in a possibly different order. Each portion of a spline lying inside an element between two spline-node or spline-edge intersections is labeled as an "embedded edge," as illustrated in Figure 4-2, and is identified by the spline arc-length parameters at the two intersections.

The orientation of the splines is used to determine the direction of validity of each cut, where a valid direction is one that points into the computational domain. This step is also performed only once as it requires floating point calculation. Based on the intersections and validity directions, new "cut edges" are constructed from intersected edges of the background mesh. Connectivity information in the form of spline-edge intersections, triangle vertices, and spline endpoints is used to stitch together the cut edges and embedded edges into loops that enclose disjoint cut regions. Specifically, for one background triangle, denote by E the set of embedded edges and cut or whole edges. Denote by I the set of spline-edge/spline-node intersections and background triangle vertices that form the endpoints for the members in E. For example, for the triangle of interest in Figure 4-2, E consists of one embedded edge, two cut edges, and one whole edge, while I consists of two spline-edge intersections and two vertices of the triangle. In general, each $i \in I$ joins two members of E, labeled by $I2E(i, 1)$ and $I2E(i, 2)$, where $I2E$ is the intersection-to-edge connectivity list. Traversing this list once, an edge-to-intersection list, $E2I$, is constructed, where $E2I(e, 1)$ and $E2I(e, 2)$ are the two intersections adjacent to an edge $e \in E$. The algorithm for creating loops enclosing disjoint cut regions proceeds as follows:

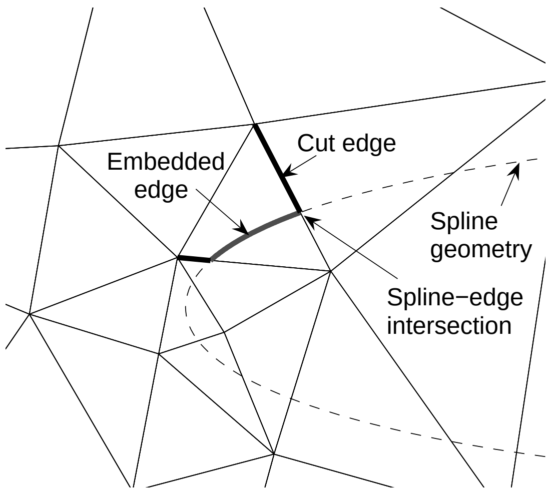


Figure 4-2: Intersection between a background mesh and an airfoil spline geometry. An embedded edge and two cut edges are identified for one triangle. Embedded edges consist of contiguous portions of the spline geometry inside the background mesh triangles, whereas cut edges consist of portions of background triangle edges inside the computational domain.

Loop creation algorithm

```
1 From $E$, pick an edge, $e_0$.

2 Begin a new loop. Let $i_0 = E2I(e_0, 1)$. Note, one could also choose $E2I(e_0, 2)$, in
  which case the order of the edges in the loop would be reversed.

3 Set $e = e_0$, $i = i_0$.

4 Do:

    – Add $e$ to the loop.
    – One of the edges, $I2E(i, 1)$ or $I2E(i, 2)$, is $e$. Set $e$ to the other edge.
    – One of the intersections, $E2I(e, 1)$ or $E2I(e, 2)$ is $i$. Set $i$ to the other intersection.

  Repeat until $e = e_0$.

5 If all edges are part of loops, the algorithm is done. Otherwise, pick a new edge, $e_0$,
  that is not yet part of a loop and return to step 2.
```

Each loop encloses a disjoint region as long as no loops are contained within other loops. The latter case only occurs when an entire piece of the geometry, consisting of one or more splines, is completely contained within a background triangle. A check for this special case is performed separately.

The enclosed disjoint regions are the newly-formed cut cells. Note that cut cells may have a nearly arbitrary number of edges and neighbors, depending on the geometry of the cuts. Figure 4-3 shows the cut cells formed near a trailing edge of an airfoil. The background triangle at the apex of the trailing edge becomes one cut cell with four neighbors, while the adjacent triangle straddling the airfoil is split into two cut cells.

In this work, disjoint regions of the cut background triangles are identified with distinct cut cells, and separate interpolation polynomials are used within each cut cell. An alternate approach, not taken in this work, is to use a single polynomial over all cut regions associated with one background triangle. For example the solution on the two disjoint regions arising from the triangle on the left in Figure 4-3 would be interpolated with a single polynomial. While this approach is simpler in that it does not require detection of disjoint regions, solution representation is adversely affected because one polynomial may not be suitable for interpolating the solution in possibly very different flow regions. This effect is especially troublesome during adaptation, leading to oscillatory convergence as multiply-cut elements are alternately over- and under-refined. Specifically, a multiply-cut element interpolating

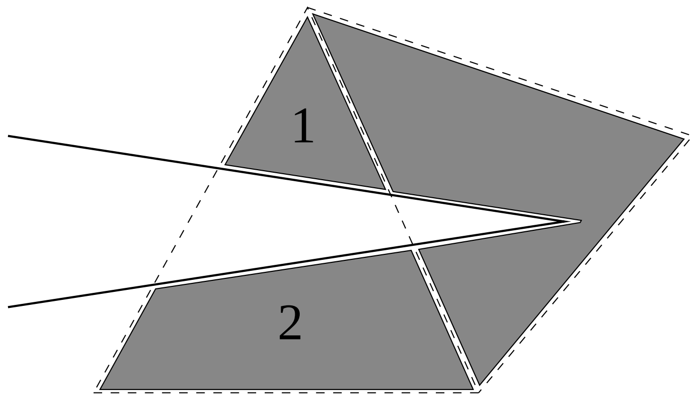


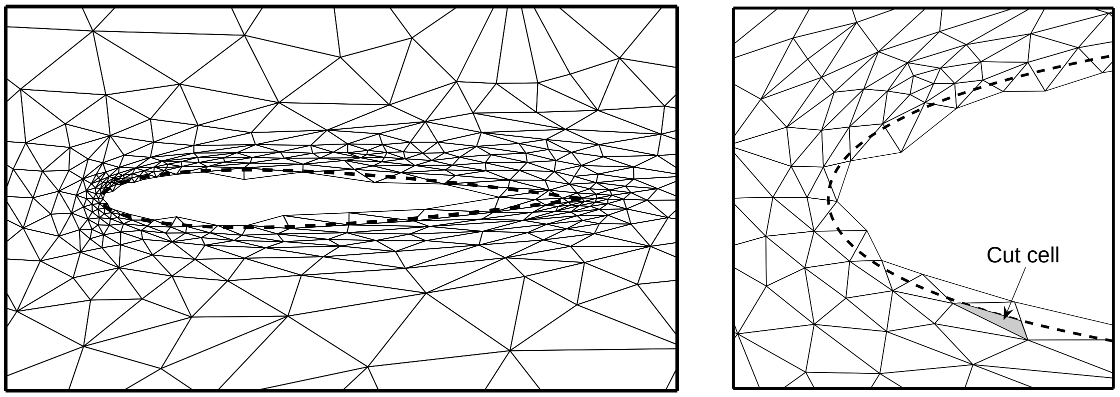
Figure 4-3: Example of cut cells formed at an airfoil trailing edge. The background triangle on the left is split into two cut cells, as indicated by the labels "1" and "2." The triangle on the right becomes one cut cell with four neighbors.

different flow fields will likely contain high error and be targeted for refinement in the subsequent adaptation iteration. On the refined mesh, the errors would drop substantially, resulting in a request for larger elements on the next adaptation iteration and again producing multiply-cut cells. In [25], this problem is alleviated by refining multiply-cut edges and triangles. While satisfactory for the cases considered, this refinement is not practical for thin, curved geometries, especially in three dimensions. Under the current approach of associating each disjoint region with an individual cut cell, this ad-hoc mesh refinement is not necessary.

During the creation of cut cells and edges, nodes of the cut background triangles are marked as lying either inside or outside the computational domain. This information is propagated to all other nodes of the background mesh by traversing the non-cut edges and triangles. Triangles contained completely within the geometry and, hence, outside the computational domain are identified according to the status of their adjacent nodes. These triangles (if any exist) are eliminated from the computational data structure such that no finite element calculations are performed on these triangles. An example of a cut-cell mesh around a NACA 0012 airfoil is shown in Figure 4-4. Triangles completely contained within the geometry are not shown. Triangles that intersect the geometry are shown in entirety, although the cut cells consist only of the portions inside the computational domain.

4.1.3 Integration

As described in the discretization (Chapter 2), a high-order DG method requires integration over element interiors as well as over element boundaries, which consist of edges inside or on the boundary of the computational domain. One-dimensional integration on



(a) Entire airfoil (b) Leading edge

Figure 4-4: Cut-cell mesh around a NACA 0012 airfoil with completely-contained triangles removed. Triangles straddling the geometry boundary yield cut cells; one cut cell is shaded in (b). The dashed line indicates the spline geometry.

cut edges and embedded edges is performed by mapping each segment to a reference interval and using numerical quadrature. Mapping of curved spline segments is done using the spline arc-length parameter, s. Figure 4-5 shows sample quadrature points for one-dimensional integration along the boundary of a cut cell. Boundary integrals often require a normal vector, which is readily available for the straight cut edges. On curved spline segments, the normal vector is in the direction $[dy/ds, -dx/ds]$, where x and y are the spline coordinates defined in equation (D.1) of Appendix D.

Currently, each spline segment of an embedded edge is mapped to a reference interval separately. In Figure 4-5, this is indicated by separate sets of Gauss points on either side of the knot in the embedded edge. This splitting at spline knots, where the geometry in general contains third-order discontinuities, leads to more accurate integration; specifically, exact quadrature for constant integrands. While useful in development, this splitting could be avoided in practice in order to reduce the computational costs of embedded-boundary integrations.

Integration on cut-cell interiors is not as straightforward as on the boundaries. One possibility is to subdivide each cut cell into possibly-curved simplices on which standard quadrature rules apply. However, this subdivision on each cut cell reduces to the original problem of meshing a domain with curved boundaries. A more general approach, used in this work, is outlined below. The goal of this method is to produce for each cut cell

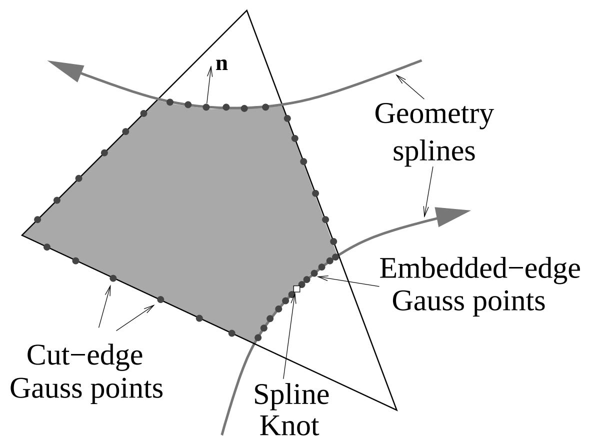


Figure 4-5: Quadrature points for one-dimensional integration on the boundary of a sample cut cell. Distinct spline segments, with endpoints at spline knots, are treated separately.

N_{quad} integration points, $\mathbf{x}_q$, and associated weights, w_q, to integrate arbitrary $f(\mathbf{x})$ via a quadrature-like sum,

$$\int_\kappa f(\mathbf{x})d\mathbf{x} \approx \sum_q w_q f(\mathbf{x}_q). \tag{4.1}$$

The key idea is to project $f(\mathbf{x})$ onto a space spanned by N_{basis} high-order basis functions, $\zeta_{\mathbf{i}}(\mathbf{x})$, where the subscript $\mathbf{i}$ indexes the basis functions. The $\zeta_{\mathbf{i}}(\mathbf{x})$ are chosen to allow for simple computation of the integral $\int_\kappa \zeta_{\mathbf{i}}(\mathbf{x})d\mathbf{x}$. In particular, choosing $\zeta_{\mathbf{i}} \equiv \partial_k G_{\mathbf{i}k}$, where $k \in [1, .., d]$, leads, by the divergence theorem, to

$$\int_\kappa \zeta_{\mathbf{i}} d\mathbf{x} = \int_\kappa \partial_k G_{\mathbf{i}k} d\mathbf{x} = \int_{\partial\kappa} G_{\mathbf{i}k} n_k ds, \tag{4.2}$$

where the n_k are components of the outward-pointing normal. Note, implied summation is used on the repeated index k. While $d = 2$ for cut cells in two dimensions, the following equations generalize naturally to three dimensions, and hence d is kept as a parameter. The integrals over the element boundary, $\partial\kappa$, appearing on the right-hand side of (4.2) are computed using the cut-edge and embedded-edge quadrature rules so that the interior integrals of the $\zeta_{\mathbf{i}}$ are calculated in a straightforward fashion. The functions $G_{\mathbf{i}k}$ are chosen as

$$G_{\mathbf{i}k}(\mathbf{x}) = x_k \Phi_{\mathbf{i}}(\mathbf{x}), \quad \Phi_{\mathbf{i}}(\mathbf{x}) = \prod_k \phi_{i_k}(x_k), \quad \mathbf{x} = [x_k], \quad \mathbf{i} = [i_k]. \tag{4.3}$$

Note, for convenience in representing tensor-product basis functions, $\mathbf{i}$ is treated as a vector

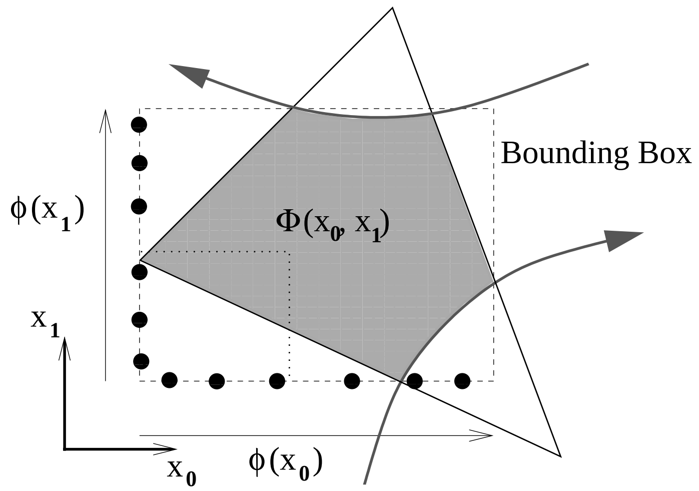


Figure 4-6: Construction of the $\Phi_{\mathbf{i}}(\mathbf{x})$ functions for use in defining the integration basis, $\zeta_{\mathbf{i}}(\mathbf{x})$. The $\Phi_{\mathbf{i}}(\mathbf{x})$ are tensor products of one-dimensional Lagrange functions, $\phi_{i_k}(x)$, in each spatial direction on the cut-cell bounding box.

index with d components, i_k. The functions $\phi_{i_k}(x)$ are well-conditioned, one-dimensional basis functions. In this work, Lagrange basis functions are used with Gauss point nodes on the element bounding box intervals (except for certain ill-conditioned cases, as described at the end of this section), as shown in Figure 4-6. The order of these functions is the desired order of integration for $f(x)$. This order depends on the equation set and on the solution interpolation order, p. The same order is used for cut cells as for standard elements; for compressible the Navier-Stokes equations, in which the integrands are not polynomial, an order of $2p + 1$ has been found to be sufficient [24]. Note, the $\zeta_{\mathbf{i}}$ and $\Phi_{\mathbf{i}}$ basis functions are used only for the calculation of cut-cell integration rules. They are different from the basis functions used for solution approximation.

The factors of x_k in the definition of $G_{\mathbf{i}}$ ensure that the basis functions $\zeta_{\mathbf{i}} = \partial_k G_{\mathbf{i}k} = d\Phi_{\mathbf{i}}(\mathbf{x}) + x_k \partial_k \Phi_{\mathbf{i}}(\mathbf{x})$ span the same complete space as the tensor product functions $\Phi_{\mathbf{i}}$. This statement can be proved as follows. Let

$$\mathbf{x}^{\mathbf{j}} \equiv \prod_k x_k^{j_k}, \quad k \in [1, .., d],$$

be the standard monomial basis indexed by $\mathbf{j} = [j_k]$. For example, $\mathbf{j} = [4, 3]$ represents the monomial $x_1^4 x_2^3$. Since the $\Phi_{\mathbf{i}}(\mathbf{x})$ are assumed to form a complete basis, every monomial can be written as a linear combination of the $\Phi_{\mathbf{i}}$,

$$\mathbf{x}^{\mathbf{j}} \;=\; a_{\mathbf{j}\mathbf{i}} \Phi_{\mathbf{i}}(\mathbf{x}),$$

where the $a_{\mathbf{ji}}$ are constants. Taking the gradient of both sides, followed by an inner product with x_k yields

$$\begin{aligned} x_k \partial_k \left(\mathbf{x}^{\mathbf{j}}\right) &= x_k \partial_k \left(a_{\mathbf{ji}} \Phi_{\mathbf{i}}(\mathbf{x})\right), \\ \left(\sum_k j_k\right) \mathbf{x}^{\mathbf{j}} &= a_{\mathbf{ji}} x_k \partial_k \Phi_{\mathbf{i}}(\mathbf{x}). \end{aligned}$$

Adding $d\,\mathbf{x}^{\mathbf{j}} = d\,a_{\mathbf{ji}}\Phi_{\mathbf{i}}$ to both sides results in

$$\left(\sum_k j_k + d\right) \mathbf{x}^{\mathbf{j}} = a_{\mathbf{ji}} \left(d\,\Phi_{\mathbf{i}}(\mathbf{x}) + x_k \partial_k \Phi_{\mathbf{i}}(\mathbf{x})\right) = a_{\mathbf{ji}} \zeta_{\mathbf{i}}(\mathbf{x}).$$

Since $d > 0$ and the j_k are non-negative, dividing both sides by $(\sum_k j_k + d)$ yields a representation of each monomial by the $\zeta_{\mathbf{i}}$ functions. Thus, the $\zeta_{\mathbf{i}}$ span the same complete space as the $\Phi_{\mathbf{i}}$.

The projection of $f(\mathbf{x})$ onto $\zeta_{\mathbf{i}}(\mathbf{x})$, $f(\mathbf{x}) \approx \sum_{\mathbf{i}} F_{\mathbf{i}} \zeta_{\mathbf{i}}(\mathbf{x})$, is performed by minimizing the sum of squares of the projection error,

$$E^2 = \sum_q \left[\sum_{\mathbf{i}} F_{\mathbf{i}} \zeta_{\mathbf{i}}(\mathbf{x}_q) - f(\mathbf{x}_q)\right]^2,$$

where q in the sum ranges over the sampling points, $\mathbf{x}_q$, and $f(\mathbf{x}_q)$ is the value of the integrand evaluated at $\mathbf{x}_q$. The solution vector, $F_{\mathbf{i}}$, is found using QR factorization of the $N_{\text{quad}} \times N_{\text{basis}}$ matrix $\zeta_{\mathbf{i}}(\mathbf{x}_q)$,

$$F_{\mathbf{i}} = (R^{-1})_{\mathbf{ij}} (Q^T)_{\mathbf{j}q} f(\mathbf{x}_q), \quad \text{where} \quad \zeta_{\mathbf{i}}(\mathbf{x}_q) = Q_{q\mathbf{j}} R_{\mathbf{ji}}.$$

Thus, the integral of $f(\mathbf{x})$ over κ can be written as

$$\int_\kappa f(\mathbf{x}) d\mathbf{x} \approx \underbrace{\sum_{\mathbf{i}} F_{\mathbf{i}} \int_\kappa \zeta_{\mathbf{i}}(\mathbf{x}) d\mathbf{x}}_{\text{projection}} = \sum_q f(\mathbf{x}_q) \underbrace{Q_{q\mathbf{j}} (R^{-T})_{\mathbf{ji}} \int_\kappa \zeta_{\mathbf{i}}(\mathbf{x}) d\mathbf{x}}_{w_q}, \tag{4.4}$$

where the expression for the sampling weights is deduced by comparing (4.4) to (4.1). Note, the summation symbol is included only for clarity; implied summation is still used for all repeated indices. In the last step in (4.4), a change in the order of summation between **i** and q was performed. None of the terms in the sampling weights expression, $Q, R, \int_\kappa \zeta_{\mathbf{i}}(\mathbf{x}) d\mathbf{x}$, depend on the integrand function, $f(\mathbf{x})$. Thus, for each cut-cell interior, the sampling weights can be computed once in a pre-processing step and used for arbitrary integrands.

The choice of sampling points, $\mathbf{x}_q$, affects the conditioning of the QR factorization of $\zeta_{\mathbf{i}}(\mathbf{x}_q)$. The points should lie inside the cut cell so that the integrand remains physical. For example, quantities such as pressure or temperature may not necessarily evaluate to a positive value outside the cut cell. Multiple methods exist for choosing these interior points. One option consists of picking points randomly in the element bounding box and checking whether or not they lie inside the element. The inside/outside check can be performed by casting a segment to a point that is known to be inside or outside the element and counting the number of intersections with the 1D element boundaries. However, this method becomes inefficient for elements whose area is small relative to the area of the bounding box, in which case a randomly-chosen point is likely to lie outside the cut cell. Thus, in this work, the random points are chosen by an alternate method, which involves casting interior-bound rays from quadrature points on the 1D element boundary, as shown in Figure 4-7. These

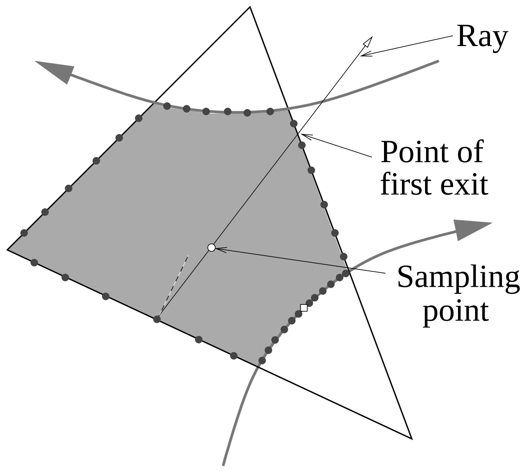


Figure 4-7: Interior sampling point selection by ray casting from boundary quadrature points. The rays are cast with a random perturbation in angle from the inward-pointing normal. For each ray, one sampling point is chosen between the ray origin and the point of first exit.

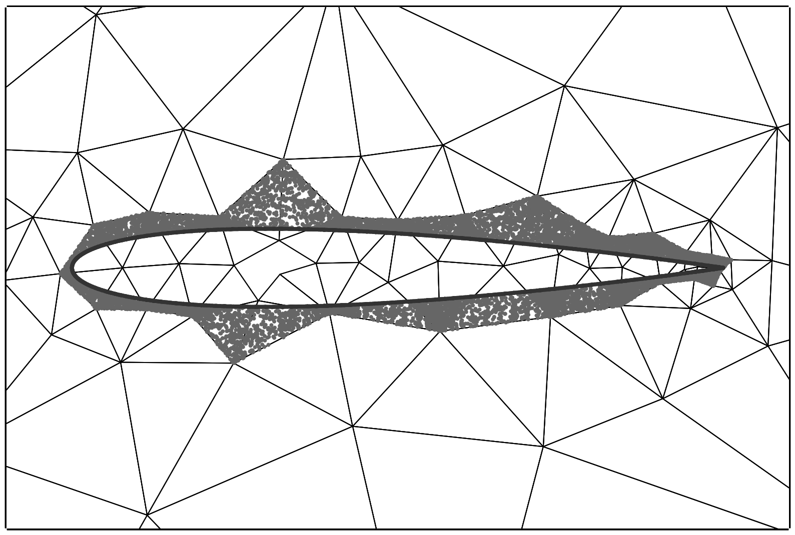
rays are directed normal to the element boundary with a random angle variation (default range is $\pm 15^o$). The closest intersection of each ray with the element boundary marks where the ray first exits the element. A random interior point is chosen between the origin of the ray and the exit point. This process repeats until the desired number of sampling points is obtained, each time using a randomly-chosen boundary quadrature point. The random choice of boundary quadrature points is weighted by the length of the boundary edges. That is, quadrature points on longer edges are more likely to be selected than points on shorter edges. This weighting is performed to avoid clustering of sampling points that could result from many rays being cast from a very short boundary edge. Example sampling points produced by the ray-casting method for a cut-cell mesh around a NACA 0012 are shown in Figure 4-8.

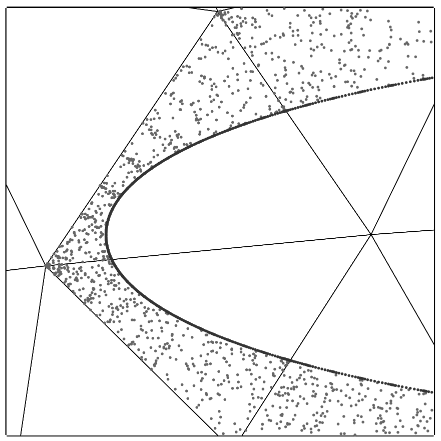
Since a random set of points may possess unfavorable clusters, conditioning of the QR factorization generally improves with an increasing number of sampling points, N_{quad}. In this work, N_{quad} is set to four times the number of $\zeta_{\mathbf{i}}$ basis functions: $N_{\text{quad}} = 4N_{\text{basis}}$. In the event of a singular error in the QR factorization, an event that has not yet been observed for any cases in practice, another set of sampling points is chosen. In addition, using an axis-aligned element bounding box to define the $\Phi_{\mathbf{i}}(\mathbf{x})$ may pose conditioning problems for non-axis-aligned sliver elements, as shown in Figure 4-9. In this case, conditioning is improved by rotating the bounding box for a tighter fit around the element. Specifically, for each element, two new bounding boxes are constructed, oriented along the diagonals of the original axis-aligned bounding box. The tightest-fitting bounding box, which is the one with the smallest area, is used.

Better algorithms likely exist for performing the cut-cell integrations or for improving the proposed method. For example, the sampling point selection process can be made more sophisticated. Random selection was used here for its simplicity, justified by the fact that unfavorable clusters are detected by degeneracy in the weight calculation, in which case the selection process is repeated. An improved selection process may be more expensive, but it may allow for fewer sampling points. This is an area of possible future research.

4.1.4 Adaptation on Cut-Cell Meshes

One of the primary advantages of using cut cells in an adaptive method lies in being able to re-mesh the domain at each iteration according to some prescribed metric without requiring the mesh to conform to an intricate geometry. Such re-meshing requires specification of the metric request everywhere in the background domain, including inside the geometry. The standard approach of specifying the metric on the nodes or elements of the current mesh must be modified for cut cells, as the cutting process removes triangles contained completely within the geometry. The approach taken in this work is to store, at each adaptation iteration, the original background mesh from which the cut-cell mesh is obtained. The metric for re-meshing is prescribed on this background mesh. In cases of multiple cut cells originating from the same background triangle, the metric on the background triangle is taken to be the component-wise average of the metrics on the multiple cut cells. As no solution exists outside the computational domain, a grid-implied metric (Section 3.2.2) is used on background-mesh triangles completely contained within the geometry. This choice preserves the sizes of these contained triangles, except possibly at the geometry boundary,


(a) Entire airfoil


(b) Leading edge

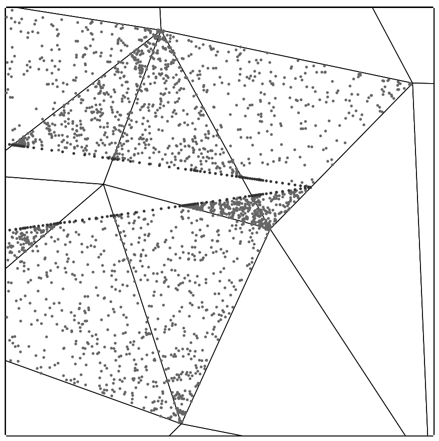

(c) Trailing edge

Figure 4-8: Example of integration points for cut cells around a NACA 0012 geometry. In addition to the cut-cell interior sampling points, quadrature points on the embedded edges are also shown.

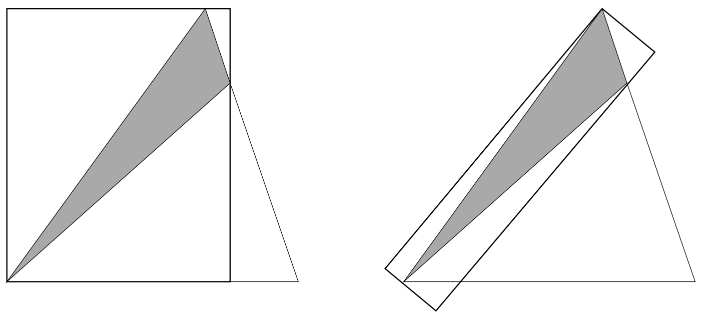


Figure 4-9: For non-axis-aligned sliver cut elements (shaded area), the original bounding box (left) is rotated to obtain a tighter fit (right). Also shown is the original triangle from which the sliver element was cut.

where metric smoothing by the mesher or by the element-to-node metric transfer brings in influence from neighboring cut cells. Such smoothing prevents any sudden changes in mesh size near the geometry boundary.

Error estimation on cut cells remains fundamentally unchanged from the boundary-conforming case. During the solution transfer from the current mesh to the adapted mesh (described in Section 3.2.3), $p = 0$ restriction is performed in lieu of L_2 projection on cut cells to prevent the creation of non-physical states. Specifically, on every cut cell in the adapted mesh, the transferred solution consists of a weighted average of the solutions from overlapping elements of the current mesh.

4.1.5 Implementation

The cut-cell method was implemented in an existing discontinuous Galerkin finite element code, with no change to the basic numerical integration paradigm on element interiors and boundaries. No changes were necessary because the cut-cell integrations take the form of quadrature sums, as indicated in (4.1) and (4.4). For each cut cell, integration rules consisting of sampling points and associated sampling weights are created once in a pre-processing step and saved before beginning solution iterations. For integration on embedded boundaries, boundary normals are also pre-computed and stored. Solution approximation functions for cut cells are defined not on the original triangles, but rather on "shadow" triangles. A shadow triangle is taken to be the right triangle obtained by dividing in half the oriented bounding boxes shown on the right in Figure 4-9. This choice is permissible because element-wise continuity in the solution approximation is not enforced for the discontinuous Galerkin method. Thus, the basis for each element may be practically arbitrary. Using shadow triangles improves conditioning of the approximation basis for small and/or sliver cut cells. The basis functions on the shadow triangles are the same as for standard elements: either nodal Lagrange or hierarchical [24].

Even though this work deals with steady-state solution via implicit schemes, an unsteady term with local time steps is used to improve robustness in the early stages of convergence. As cut cells may have very small areas, global time stepping (i.e. constant time step Δt) is not used. Rather, a local time step is chosen for each element using a global CFL number: $\Delta t_\kappa = \text{CFL}(h_\kappa / s_{\kappa,\max})$, where h_κ and $s_{\kappa,\max}$ are the element-specific size (e.g. hydraulic diameter) and maximum wave speed, respectively. The method for setting the CFL number proceeds as follows:

- At the beginning of a run, the global CFL number is initialized to CFL_0.
- Following each Newton iteration:
  - If a full state update is taken, the CFL number is increased: $\text{CFL} = \text{CFL} * \text{CFL}_{\text{increase}}$.
  - If the state update is limited, in practice, by allowable changes in pressure and density [24], the CFL number is not increased.
  - In case of a limited update, the update is taken if the limited update is greater than a prescribed fraction (default is 1%) of the original update. Otherwise, the update is not taken and the CFL number is decreased: $\text{CFL} = \text{CFL}/\text{CFL}_{\text{decrease}}$.

The parameters used in this work are $\text{CFL}_0 = 1$, $\text{CFL}_{\text{increase}} = 2$, and $\text{CFL}_{\text{decrease}} = 10$.

4.2 Results

The adaptation scheme is applied to several representative aerodynamic cases using orders $p = 1$ to $p = 3$. Comparisons of the adapted meshes and the error convergence histories are given for both boundary-conforming and cut-cell meshes in terms of degrees of freedom (DOF). The number of degrees of freedom in a solution is computed as the total number of unknowns excluding the equation-specific multiplier (e.g. 4 for the 2D Euler or Navier-Stokes equations).

For comparing different interpolation orders, p, a better metric than degrees of freedom is computational time. However, relative run times depend heavily on the implementation and on the hardware. Nevertheless, a more accurate estimate of the computational work is possible by assuming a work expression of the form, $W \sim N_e[n(p)]^a$, where N_e is the number of elements, $n(p)$ is the degrees of freedom per element, and a is a measure of the computational complexity per element. Using $\text{DOF} = N_e n(p)$, the work estimate may be written as $W \sim \text{DOF}[n(p)]^{a-1}$. From experience, at least for orders up to $p = 3$, the work is dominated by the matrix-vector products during the GMRES linear solve; i.e. $a \sim 2$.

In addition, in two dimensions, the number of degrees of freedom per element is given by $n(p) = (p+1)(p+2)/2$. Thus, as $n(1) = 3$, $n(2) = 6$, and $n(3) = 10$, for the same DOF, $p = 2$ is expected to be twice as expensive as $p = 1$, while $p = 3$ is expected to be over three times more expensive than $p = 1$.

4.2.1 Inviscid NACA 0012, $M_\infty = 0.5$, $\alpha = 2^o$

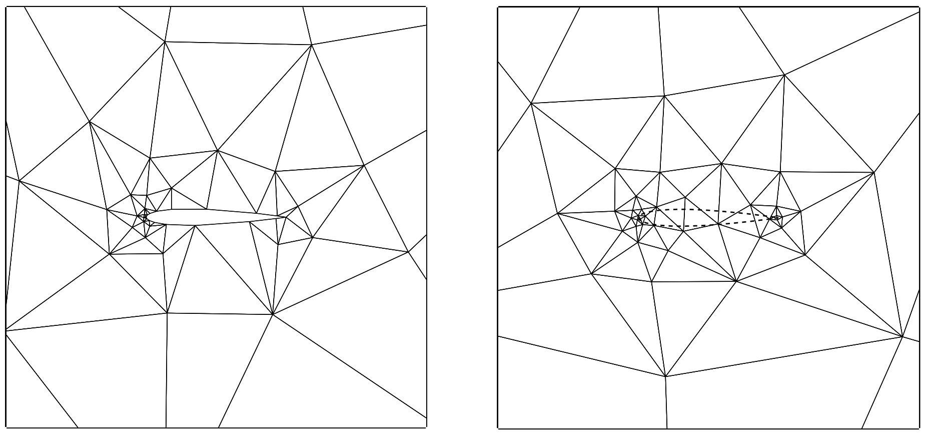
This case considers a NACA 0012 airfoil contained within a farfield box that is a distance of 100 chord lengths away from the airfoil. The NACA geometry is modified to close the trailing edge gap,

$$y = \pm 0.6(0.2969\sqrt{x} - 0.1260x - 0.3516x^2 + 0.2843x^3 - 0.1036x^4). \tag{4.5}$$

The performance of the isotropic adaptation algorithm is tested using drag as the output with a tolerance of 0.1 drag counts. As mentioned in Section 1.1, a "count" in a force calculation refers to a fraction of the corresponding non-dimensional force coefficient. Specifically, one drag count corresponds to $10^{-4}C_D$, where C_D is the drag coefficient. All two-dimensional force coefficients in this work are normalized by the freestream dynamic pressure and the chord length. In this inviscid case, the drag is obtained from the static pressure distribution on the airfoil surface with the pressure calculated using only the tangential velocity, $\mathbf{v}_t$: $p_s = (\gamma - 1)(\rho E - 0.5\rho|\mathbf{v}_t|^2)$. The "exact" output value is taken as the drag computed on a $p = 3$ boundary-conforming run adapted to 10^{-3} drag counts. Note, boundary effects of the finite farfield contribute to a nonzero drag value at steady state.

Figure 4-10 shows the initial 123-element boundary-conforming mesh and the initial 133-element cut-cell mesh. In the boundary-conforming mesh, elements adjacent to the airfoil surface are represented using cubic ($q = 3$) curved elements. Each cubic element is obtained by mapping ten Lagrange nodes from a reference right triangle to physical space. Curved edges are obtained by placing the additional edge nodes on the curved geometry. The position of the single interior node is determined by an average of the edge node positions. These boundary elements have to be curved at every adaptation iteration, since BAMG produces linear meshes. For the isotropic elements in this case, this curving does not pose a problem. In the cut-cell cases, the cubic-spline geometry contains 200 knots placed using curvature-based spacing.

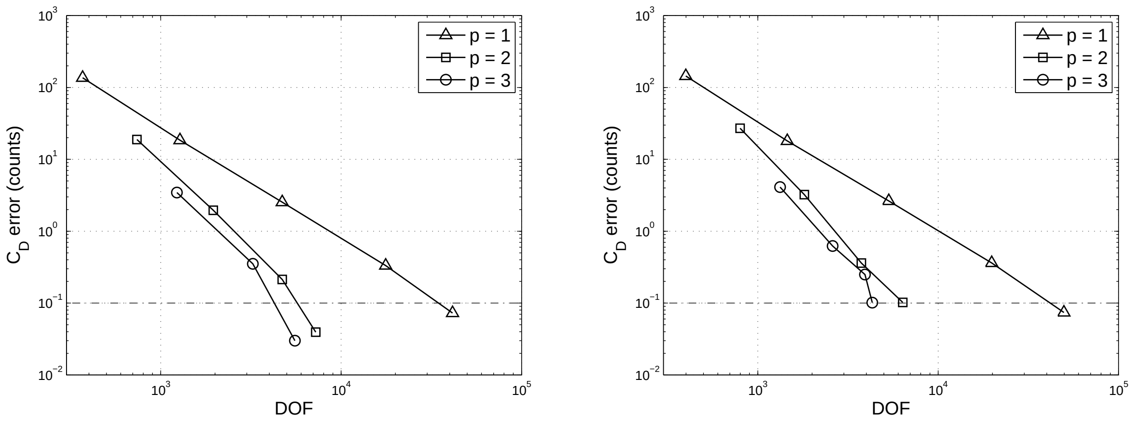
Figure 4-11 summarizes the results of the adaptation runs. The plots show the output error versus DOF at each adaptation iteration for every run. The horizontal dashed lines in both plots mark the error tolerance of 0.1 drag counts. In both the boundary-conforming and the cut-cell methods, the $p = 3$ runs achieve the desired accuracy with the fewest



(a) Boundary-conforming: 123 elements (b) Cut-cell: 133 elements

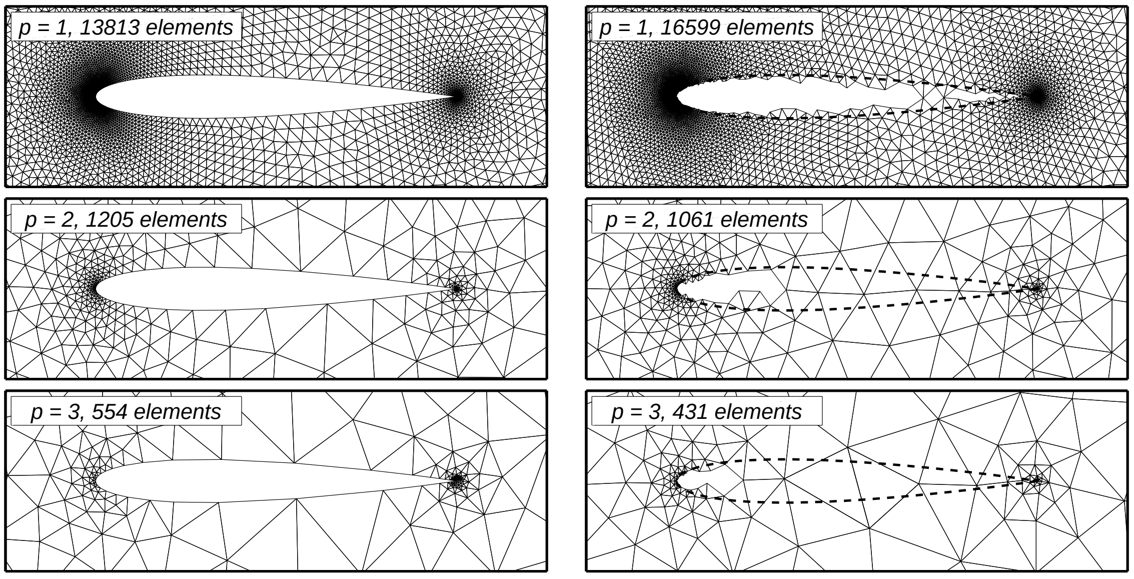
Figure 4-10: NACA 0012: $M_\infty = 0.5$, inviscid, $\alpha = 2^o$ Initial meshes for adaptive runs.

degrees of freedom. The advantage of $p = 3$ is about a factor of 1.5 over $p = 2$ and a factor of 10 over $p = 1$. In terms of the computational work estimate discussed at the beginning of this section, the differences are diminished: $p = 3$ is comparable to $p = 2$, which is almost three times less expensive than $p = 1$. A more important result, however, is that the convergence of the cut-cell runs is comparable to that of the boundary-conforming



(a) Boundary-conforming (b) Cut-cell

Figure 4-11: NACA 0012: $M_\infty = 0.5$, inviscid, $\alpha = 2^o$. Drag error versus degrees of freedom. Dashed line indicates prescribed tolerance of $e_0 = 0.1$ counts.



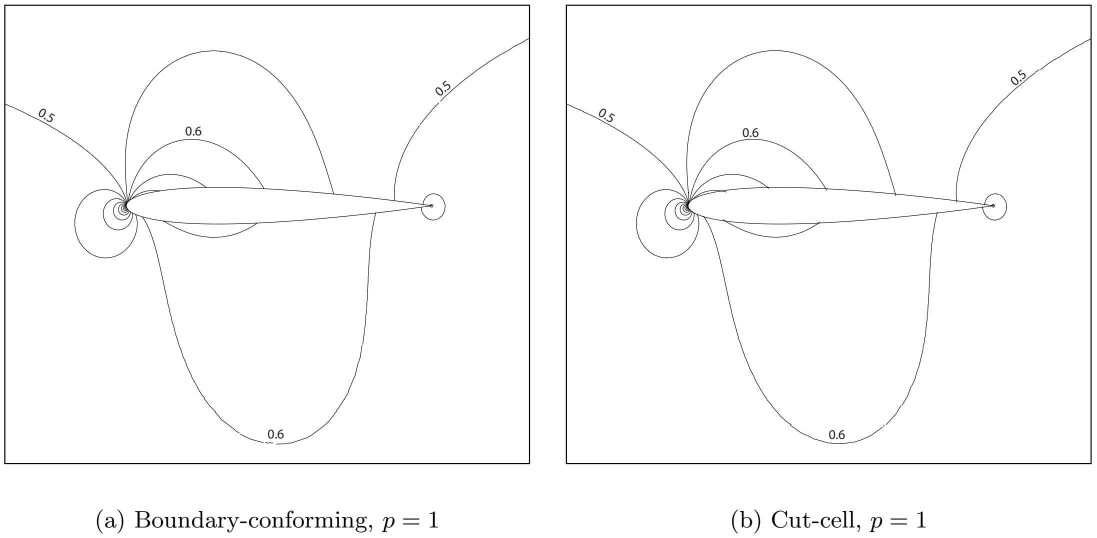
(a) Boundary-conforming (b) Cut-cell

Figure 4-12: NACA 0012: $M_\infty = 0.5$, inviscid, $\alpha = 2^o$. Final boundary-conforming and cut-cell meshes for $p = 1, 2, 3$, adapted to a drag tolerance of $e_0 = 0.1$ counts.

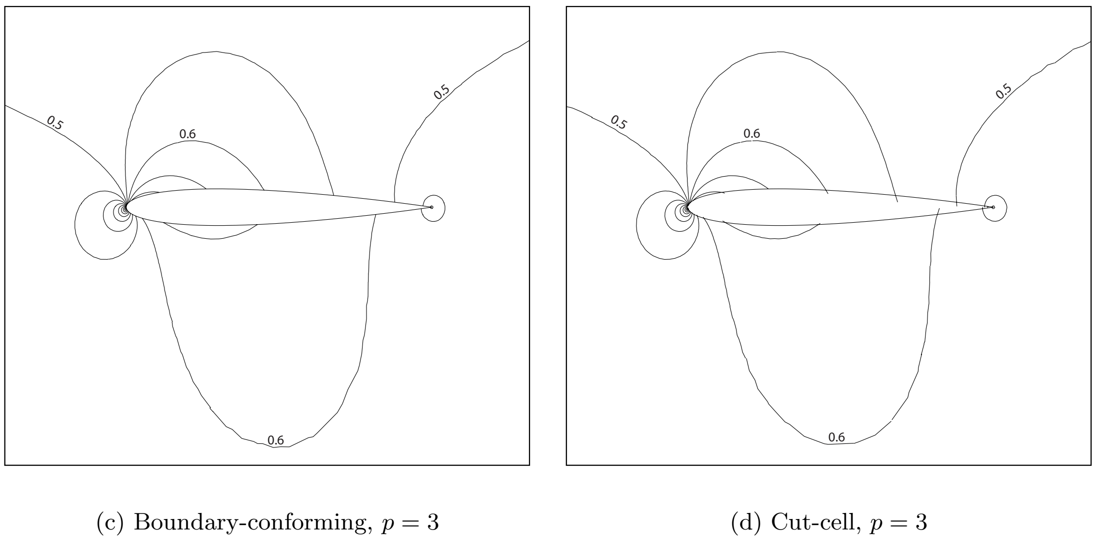
runs. This fact implies not only that the cut-cell runs converge to the same answer as the boundary-conforming runs, but also that the adaptive method is not significantly affected by the presence of cut cells.

Figure 4-12 shows close-ups of the final, adapted meshes for $p = 1, 2, 3$. The similarity between the cut-cell and boundary-conforming meshes at each p is evident, not only in the number of elements but also in their distribution in areas targeted for refinement. These areas include the leading and trailing edges, a fact that is expected as these are the areas that most significantly influence the drag output. The mesh coarseness allowed by higher-order interpolation is clear: the $p = 3$ meshes are about 30-40 times coarser than the $p = 1$ meshes.

Mach number contours for $p = 1$ and $p = 3$ on the final adapted meshes are shown in Figure 4-13. The cut-cell solutions are rendered on the background mesh triangles, and, hence, some of the contours cross the geometry boundary. However, the solution is physically valid only inside the computational domain. The contours for the four cases shown are nearly identical. As the requested error tolerance is quite strict, the flow is well resolved. In particular, no spurious dissipation wake is present.



(a) Boundary-conforming, $p = 1$ (b) Cut-cell, $p = 1$



(c) Boundary-conforming, $p = 3$ (d) Cut-cell, $p = 3$

Figure 4-13: NACA 0012: $M_\infty = 0.5$, inviscid, $\alpha = 2^o$. Mach number contours for $p = 1$ and $p = 3$ on the final adapted boundary-conforming and cut-cell meshes.

4.2.2 NACA 0012, $M_\infty = 0.5$, $Re = 5000$, $\alpha = 2^o$

In this case, a Navier-Stokes solution is computed around a NACA 0012 in a freestream Mach number of 0.5, Reynolds number ($Re = u_\infty c/\nu$) of 5000, and angle of attack of 2^o. The initial meshes are isotropic and adapted to the geometry with roughly 250 elements. The farfield is a square, 100 chord lengths away from the airfoil. Mesh optimization is performed with anisotropic elements to efficiently resolve the boundary layer and wake. In the presence of anisotropic elements near the airfoil boundary, the boundary-curving step in post-processing the linear boundary-conforming meshes is prone to failure. That is, the curved boundary may intersect interior edges and lead to un-allowable elements. This mode of failure was observed for some of the runs. One possible fix in this situation is to curve interior edges in addition to the boundary edges. However, this approach is difficult to extend to three dimensions. Therefore, when such a failure occurred, the adaptation was re-run with slightly-perturbed values for adaptation aggressiveness.

Viscous Force Calculation

A force output for a viscous simulation consists of two components: a pressure force and a viscous shear force, $\mathbf{f}^v$. The pressure force is calculated as in the inviscid example, using only the tangential velocity at the boundary. The viscous force, $\mathbf{f}^v = [f_1^v, f_2^v]$ is obtained from the viscous flux, F_{ki}^v, with a dual-consistent correction,

$$\begin{bmatrix} f_1^v \\ f_2^v \end{bmatrix} = \sum_{\sigma^{bf} \in \Gamma_{\text{airfoil}}} \int_{\sigma^{bf}} \begin{bmatrix} -F_{2i}^v n_i + \eta^{bf} \delta_{2i}^{bf} n_i \\ -F_{3i}^v n_i + \eta^{bf} \delta_{3i}^{bf} n_i \end{bmatrix} ds. \tag{4.6}$$

F_{2i}^v and F_{3i}^v are viscous momentum flux components that account for the shear stress, as presented in Chapter 2. The n_i are components of the normal vector pointing outward from the computational domain, and summation is implied on $i \in [1,..,2]$. The integral is performed over boundary edges, σ^{bf}, that lie on the airfoil boundary, Γ_{airfoil}. δ_{2i}^{bf} and δ_{3i}^{bf} are components of the auxiliary variable associated with the BR2 discretization of the viscous flux terms. Not including this correction leads to an adjoint solution that is not well-behaved at the airfoil boundary and to a loss of accuracy in the force estimate [48].

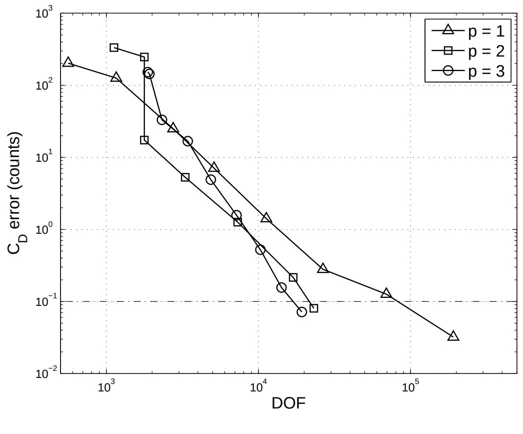
Drag Adaptation

The adaptation algorithm was first tested using drag as the output, with a tolerance of 0.1 counts. The "true" drag of 568.84 counts ($C_D = .056884$) was computed on a $p = 3$ cut-cell mesh, adapted to an error of 10^{-3} counts. The boundary-conforming and cut-cell runs converge to the same drag value. The error convergence histories from the adaptation are plotted in Figure 4-14.

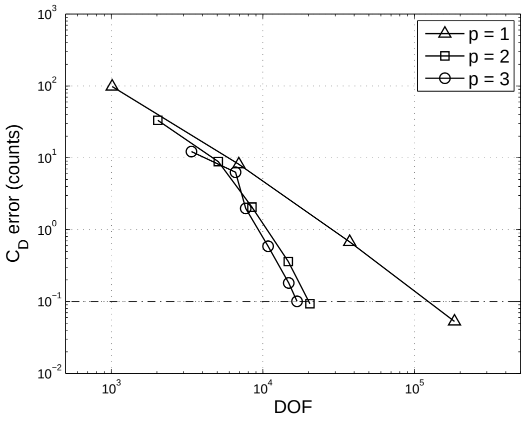
Overall, the cut-cell and boundary-conforming results are similar. For both the boundary-conforming and the cut-cell cases, $p = 3$ requires the fewest degrees of freedom at the error tolerance, although $p = 2$ does not require much more. Thus, in terms of estimated work, $p = 2$ becomes slightly advantageous to $p = 3$ in this case. $p = 1$, however, remains the most expensive, requiring a factor of 4-5 more degrees of freedom than $p = 2$, which translates to an estimated work increase of about a factor of 2.

Figures 4-15 and 4-16 show the final adapted meshes for $p = 2$ and $p = 3$, respectively. The final adapted meshes for $p = 1$ are much finer: 98808 elements for the boundary-conforming case and 61163 for the cut-cell case. They are not shown here because, on the scale used, the elements are practically indiscernible in regions of refinement. In all meshes, areas of high refinement include the boundary layer, a large extent of the wake, and, to a lesser extent, the flow in front of the airfoil. Elements in the boundary layer and in the wake are stretched in the flow direction, correctly capturing the anisotropy in this viscous solution. The similarity in element size and anisotropic stretching between the cut-cell meshes and their boundary-conforming counterparts is evident.

Mach number contours for $p = 1$ and $p = 3$ solutions on the final adapted meshes are shown in Figure 4-17. In all cases the flow appears to be well-resolved, yielding smooth contours that are nearly identical among the four plots. Mach number anisotropy is evident

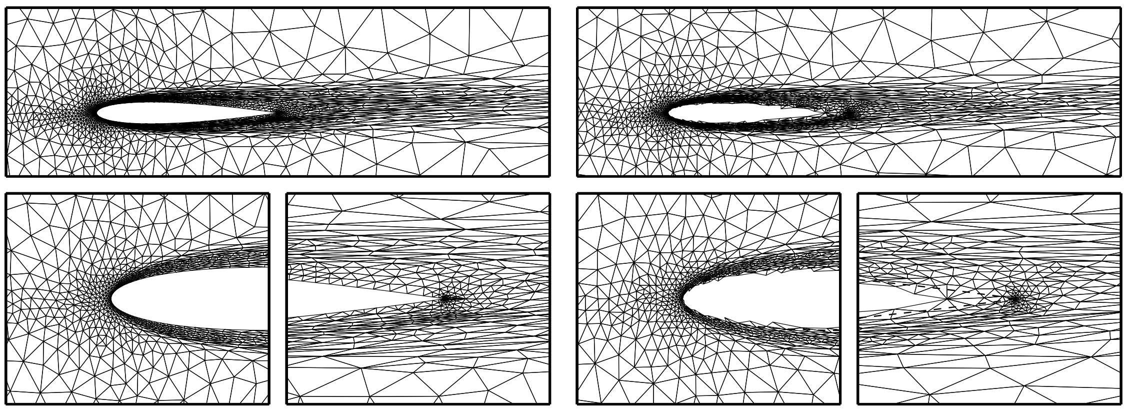


(a) Boundary-conforming



(b) Cut-cell

Figure 4-14: NACA 0012: $M_\infty = 0.5, Re = 5000, \alpha = 2^o$. Drag error versus degrees of freedom. Dashed line indicates prescribed tolerance of $e_0 = 0.1$ drag counts.

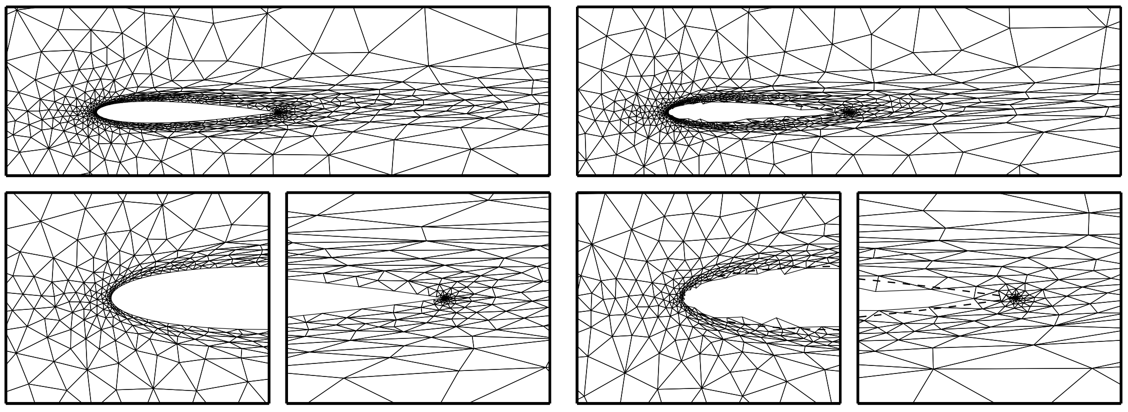


(a) Boundary-conforming: 4068 elements (b) Cut-cell: 3403 elements

Figure 4-15: NACA 0012: $M_\infty = 0.5, Re = 5000, \alpha = 2^o$. Final $p = 2$ meshes adapted on drag with tolerance $e_0 = 0.1$ counts.

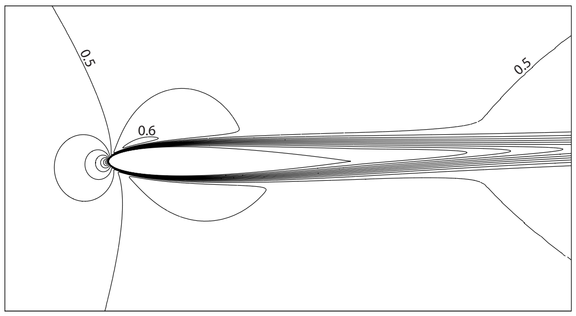
in the boundary layer and in the wake: the variation of the Mach number in the streamwise direction is much smaller than the variation in the normal direction. This anisotropy is responsible for the stretched elements in the boundary layer and in the wake.

The skin friction coefficient distributions for solutions on the final adapted meshes are shown in Figure 4-18. The skin friction coefficient, C_f, on the airfoil surface is given by $C_f = \mathbf{f}^v \cdot \mathbf{t} / \left(\frac{1}{2}\rho_\infty V_\infty^2\right)$, where $\mathbf{f}^v$ is the viscous force, $\mathbf{t}$ is a unit tangent vector along the

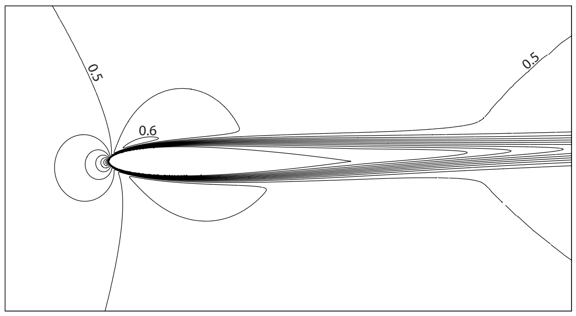


(a) Boundary-conforming: 1781 elements (b) Cut-cell: 1680 elements

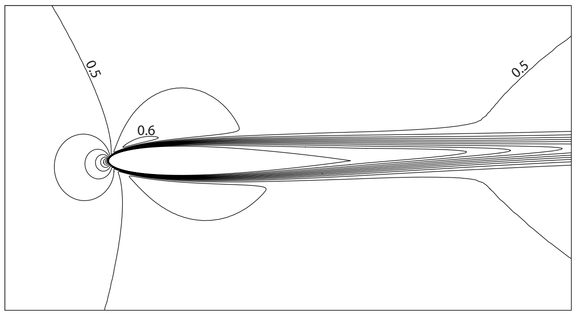
Figure 4-16: NACA 0012: $M_\infty = 0.5, Re = 5000, \alpha = 2^o$. Final $p = 3$ meshes adapted on drag with tolerance $e_0 = 0.1$ counts.



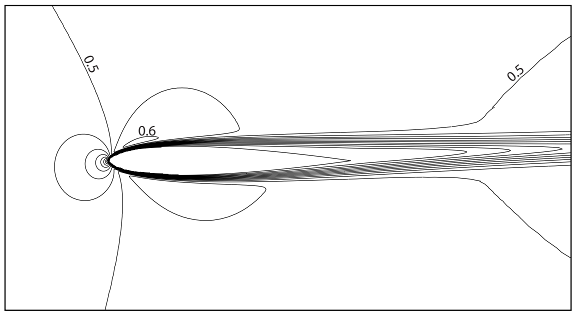
(a) Boundary-conforming, $p = 1$



(b) Cut-cell, $p = 1$



(c) Boundary-conforming, $p = 3$



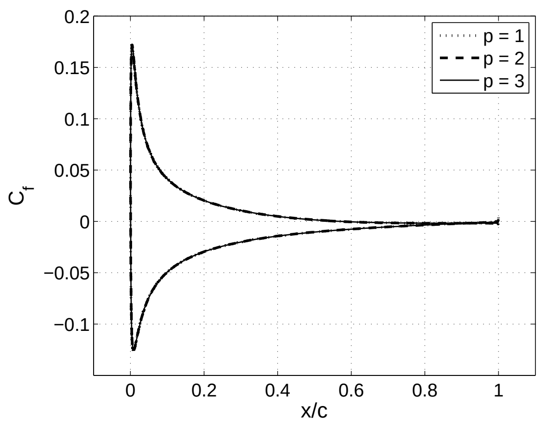
(d) Cut-cell, $p = 3$

Figure 4-17: NACA 0012: $M_\infty = 0.5, Re = 5000, \alpha = 2^o$. Mach number contours for $p = 1$ and $p = 3$ on the final adapted boundary-conforming and cut-cell meshes.

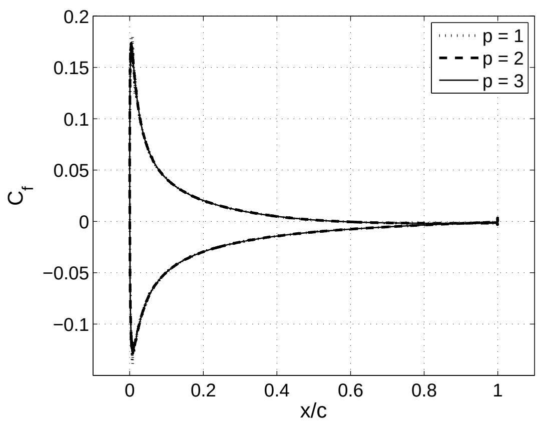
airfoil, and ρ_∞ and V_∞ are the freestream density and velocity magnitude, respectively. The flow is attached throughout; on the lower surface of the airfoil, $\mathbf{t}$ points opposite to the flow direction and C_f is negative by the above definition. As shown, the C_f plots are practically identical for the boundary-conforming and the cut-cell cases and for the different orders, p.

Lift Adaptation

A second test of the adaptation algorithm was performed using lift as the output. Each test case was adapted to 1 count of lift. The "true" lift of 369.4 counts ($C_L = .03694$). was taken from a $p = 3$ solution on a cut-cell mesh, adapted to 0.01 counts. The lift output error, relative to the true value, is shown for all adaptation runs in Figure 4-19. All runs converged to error levels below the requested tolerance. The required degrees of freedom at error tolerance are similar between the boundary-conforming and the cut-cell runs. For both methods, $p = 1$ adaptation is the least efficient in terms of error per degrees of freedom, requiring about a factor of 4 times more DOF compared to $p = 2$ at the error tolerance. In



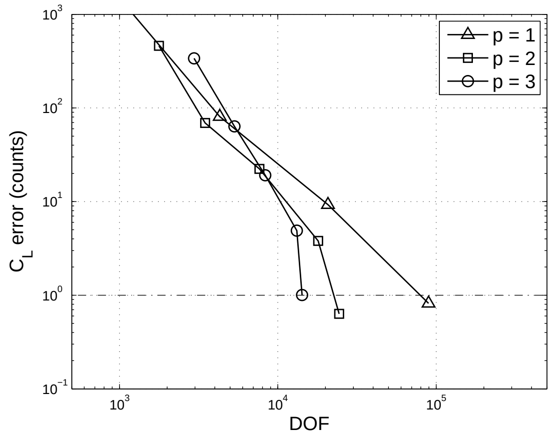
(a) Boundary-conforming



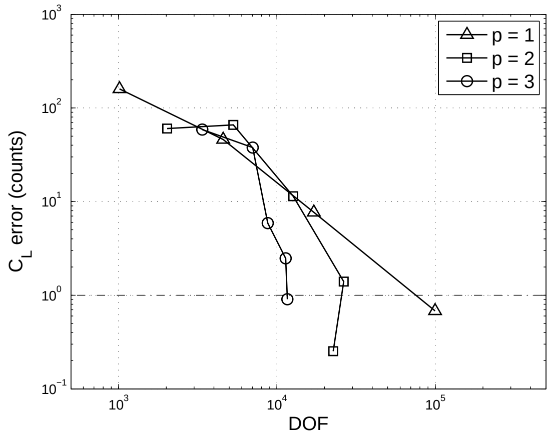
(b) Cut-cell

Figure 4-18: NACA 0012: $M_\infty = 0.5, Re = 5000, \alpha = 2^o$. Surface skin friction coefficient distributions for solutions on the final adapted boundary-conforming and cut-cell meshes. These plots were generated by evaluating C_f at the quadrature points on the embedded boundary edges.

terms of estimated work, this difference decreases to roughly a factor of 2. $p = 3$ is slightly more advantageous compared to $p = 2$ in terms of DOF; in terms of estimated work, the difference is negligible. The final lift-adapted meshes are shown in Figures 4-20 and 4-21



(a) Boundary-conforming



(b) Cut-cell

Figure 4-19: NACA 0012: $M_\infty = 0.5, Re = 5000, \alpha = 2^o$. Lift error versus degrees of freedom. Dashed line indicates prescribed tolerance of $e_0 = 1$ lift counts.

for $p = 2$ and $p = 3$, respectively. For comparison, the final $p = 1$ meshes contained 29838 elements in the boundary-conforming case and 33243 elements in the cut-cell case. The areas targeted for refinement are similar to the drag-adaptation case: the boundary layer, the leading edge, and portions of the wake. Anisotropy is clearly present in both the $p = 2$ and the $p = 3$ meshes.

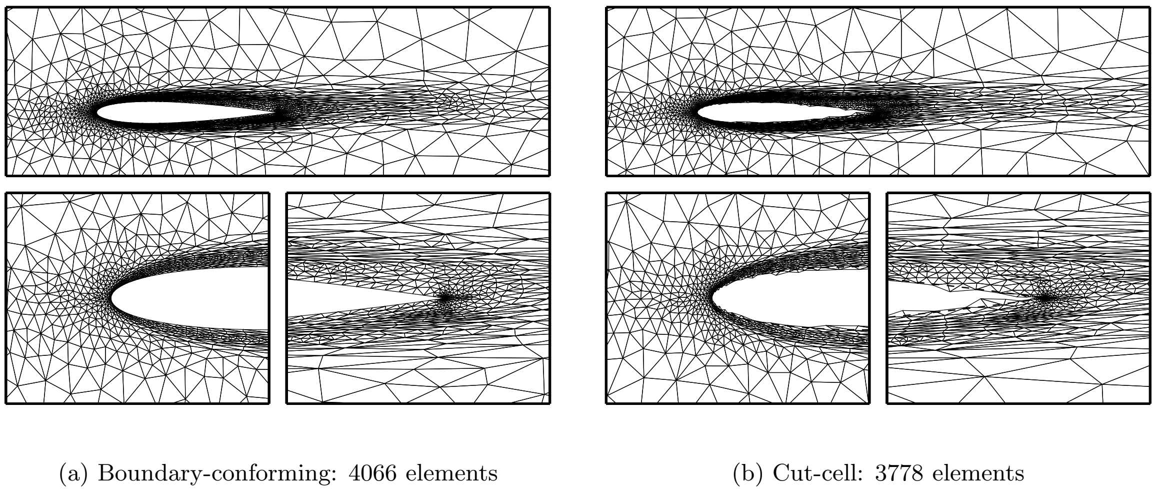

(a) Boundary-conforming: 4066 elements (b) Cut-cell: 3778 elements

Figure 4-20: NACA 0012: $M_\infty = 0.5, Re = 5000, \alpha = 2^o$. Final $p = 2$ meshes adapted on lift with tolerance $e_0 = 1$ count.

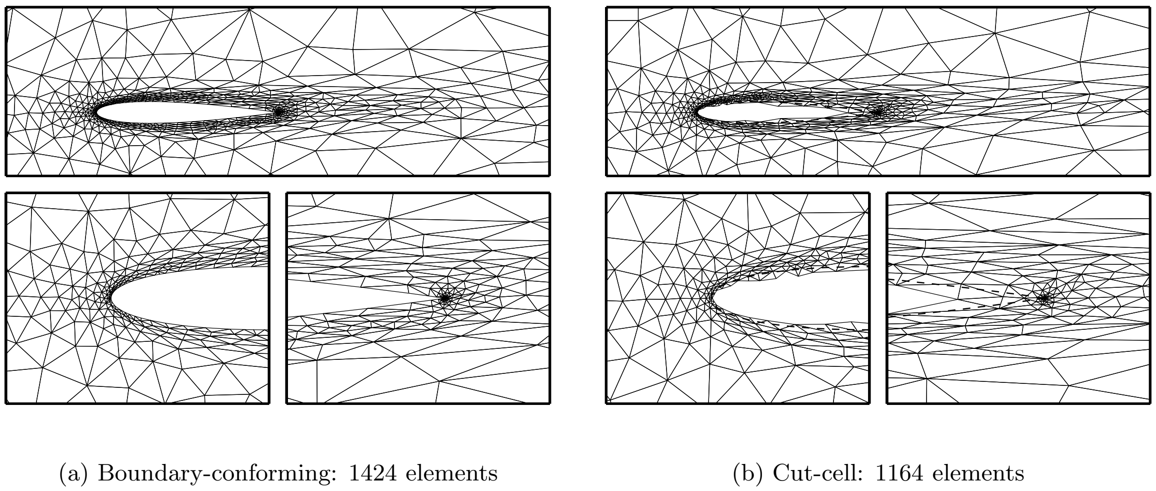
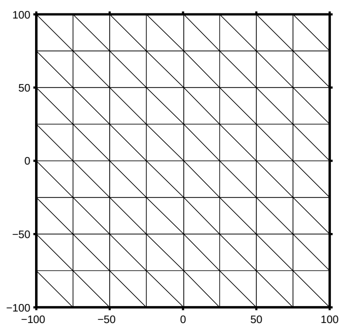

(a) Boundary-conforming: 1424 elements (b) Cut-cell: 1164 elements

Figure 4-21: NACA 0012: $M_\infty = 0.5, Re = 5000, \alpha = 2^o$. Final $p = 3$ meshes adapted on lift with tolerance $e_0 = 1$ count.

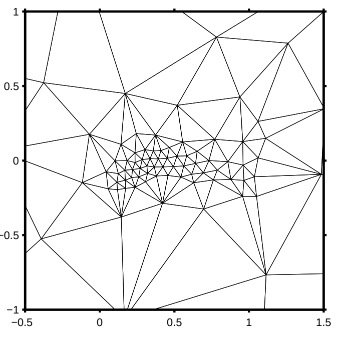
4.2.3 Sensitivity to Initial Mesh

For the adaptation method to be practical, the final adapted meshes should not be highly sensitive to the starting meshes. The sensitivity was tested for a NACA 0012 at $Re = 5000$, $M = 0.5$, $\alpha = 2^o$, adapted to drag with an error tolerance of 1 drag count. Runs were performed with several different cut-cell starting meshes, including a set of uniform, structured, triangulations of the entire domain, as well as two meshes adapted to the geometry to different levels of refinement. These meshes are shown in Figure 4-22.

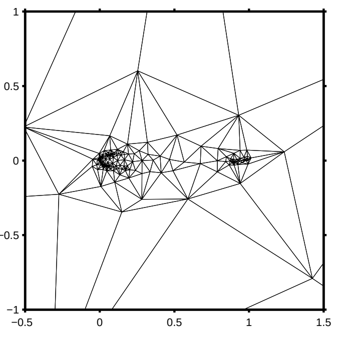
Figure 4-23 shows the adaptation histories for $p = 1, 2, 3$. For the finer uniform starting meshes, the degrees of freedom decrease rapidly in the first adaptation iteration due to coarsening of the mesh away from the airfoil, where the mesh is initially relatively too fine. The adaptation histories appear somewhat scattered for the first several iterations, but then converge as the error decreases. For a given p, the final adapted meshes are close not only in DOF count, but also in DOF spatial distribution. This observation is made by qualitatively comparing locations of refinement and element aspect ratio. In particular, Figure 4-24 shows the final adapted meshes obtained for $p = 3$, starting from the initial meshes in Figure 4-22. Although the meshes are not identical, they exhibit similar resolution near the leading and trailing edges and in the wake. Therefore, these runs illustrate that for a low-enough error tolerance, the final meshes generated by the adaptation algorithm are relatively insensitive to the initial mesh.



(a) Uniform 8x8

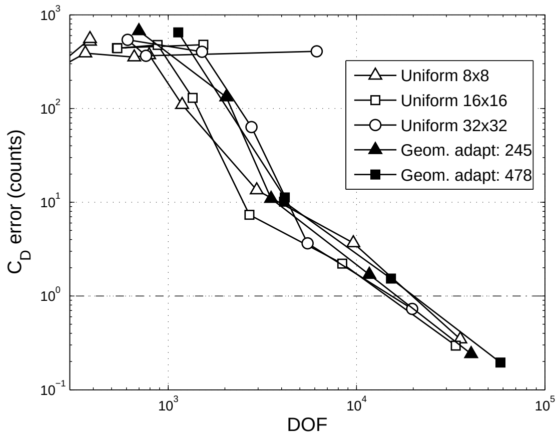


(b) Adapted to geometry: 245 elements

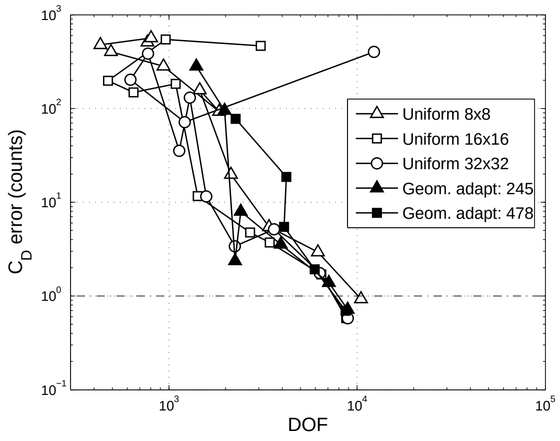


(c) Adapted to geometry: 478 elements

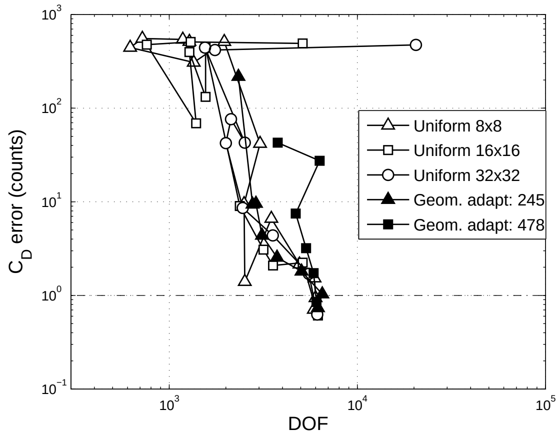
Figure 4-22: Mesh-sensitivity study: sample initial meshes showing a uniform triangulation as well as two meshes adapted to the geometry.



(a) $p = 1$



(b) $p = 2$



(c) $p = 3$

Figure 4-23: Mesh-sensitivity study: adaptation histories for $p = 1, 2, 3$, starting from various initial meshes, some shown in Figure 4-22.

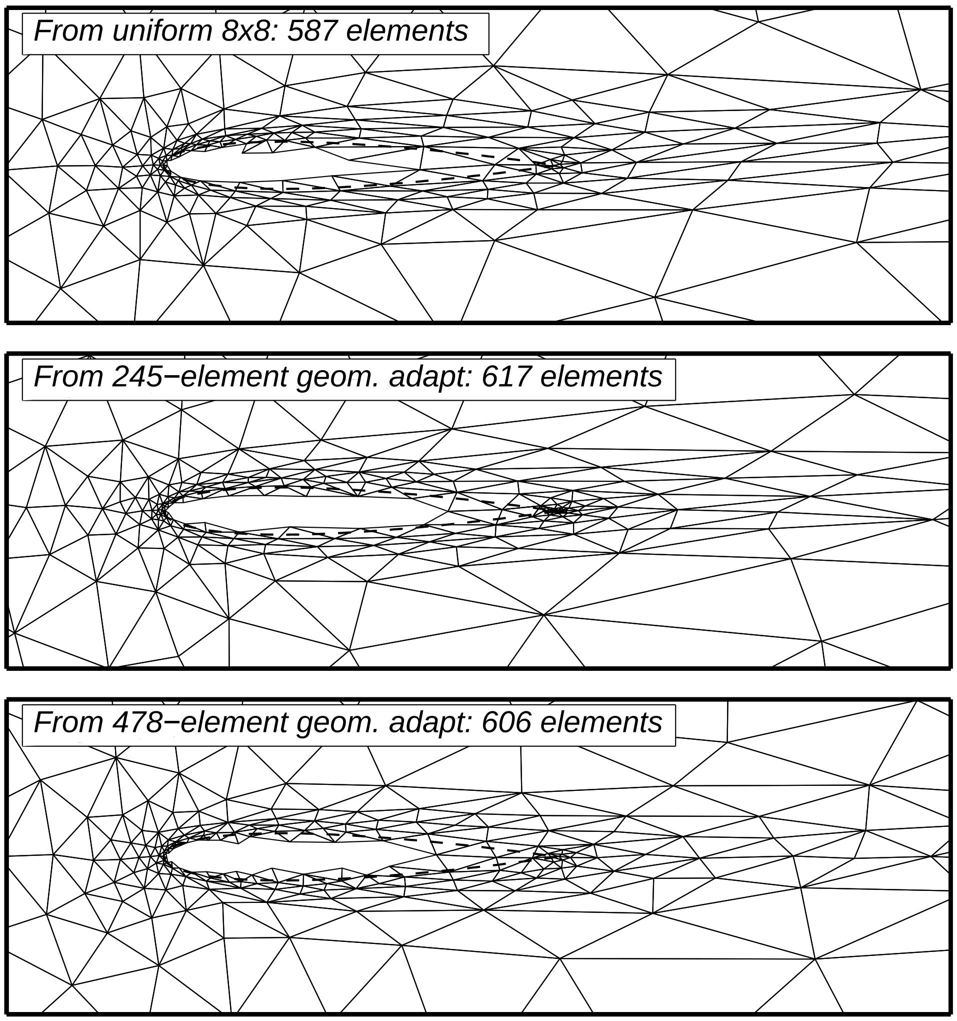

From uniform 8x8: 587 elements
From 245−element geom. adapt: 617 elements
From 478−element geom. adapt: 606 elements

Figure 4-24: Mesh-sensitivity study: final $p = 3$ meshes for the three initial meshes shown in Figure 4-22.

4.2.4 NACA 0005, $M = 0.4$, $Re = 50000$, $\alpha = 0^o$

This case considers a thin airfoil in a subsonic, high-Reynolds number flow. The NACA 0005 geometry is given by (4.5) with 0.25 for the leading coefficient instead of 0.6. The farfield is located 100 chord lengths away from the airfoil. Only cut-cell meshes are presented for this case, as boundary-conforming meshes readily yielded invalid elements upon curving the highly-stretched boundary elements in latter stages of adaptation. The initial mesh for this case, shown in Figure 4-25, was created by starting from a geometry-adapted mesh and adapting several times at $p = 2$, $Re = 10000$ to a loose drag error requirement. The resulting 677-element mesh is relatively coarse but contains adequate resolution to allow for initial $p = 2$ and $p = 3$ solves at $Re = 50000$.

The adaptation algorithm was tested using drag as the output with a tolerance of 0.01 counts. The "true" drag of 161.043 counts was taken from a $p = 3$ solution on a mesh obtained by uniformly refining the final adapted $p = 2$ mesh. The drag error histories for $p = 1, 2, 3$ are shown in Figure 4-26. At the error tolerance, $p = 3$ requires about 1.5 times fewer DOF than $p = 2$ and about a factor of 10 fewer DOF than $p = 1$. Thus, in terms of estimated work, $p = 2$ and $p = 3$ are relatively comparable, while $p = 1$ is a factor of three times more expensive.

The final adapted meshes are shown in Figure 4-27 for $p = 2$ and $p = 3$. For $p = 1$, the final adapted mesh contains 106201 elements, which would not be discernable on the scale used. Clearly, the $p = 2$ and $p = 3$ meshes are much coarser with 5859 and 2004 elements, respectively. Highly-anisotropic elements are present in the boundary layer and in the wake. The high anisotropy near the boundary makes the boundary-conforming meshes prone to

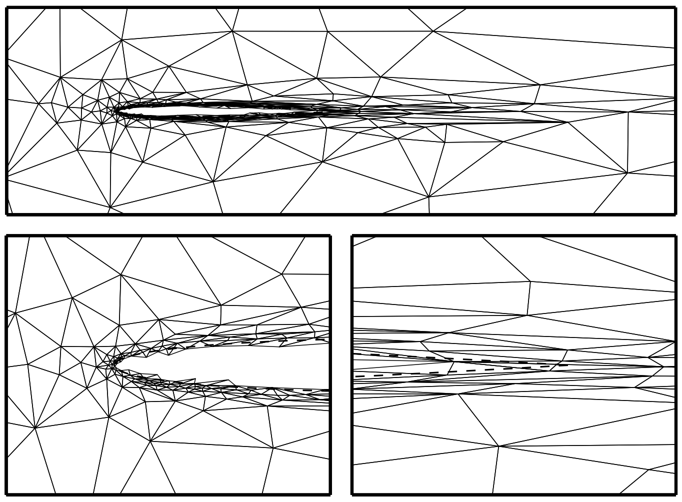


Figure 4-25: NACA 0005: $M = 0.4$, $Re = 50000$, $\alpha = 0^o$. Initial mesh consisting of 677 elements.

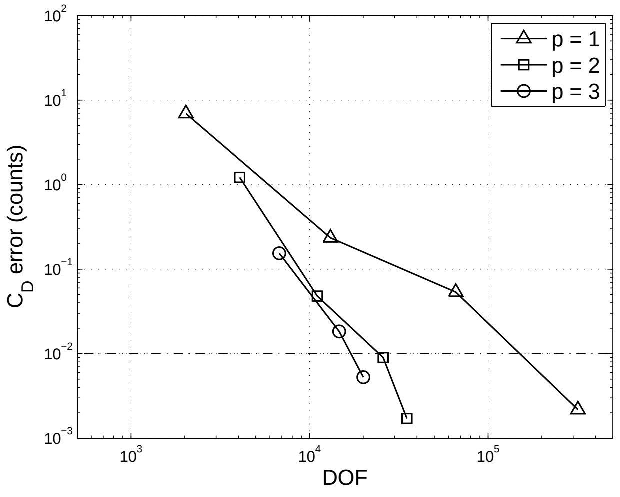
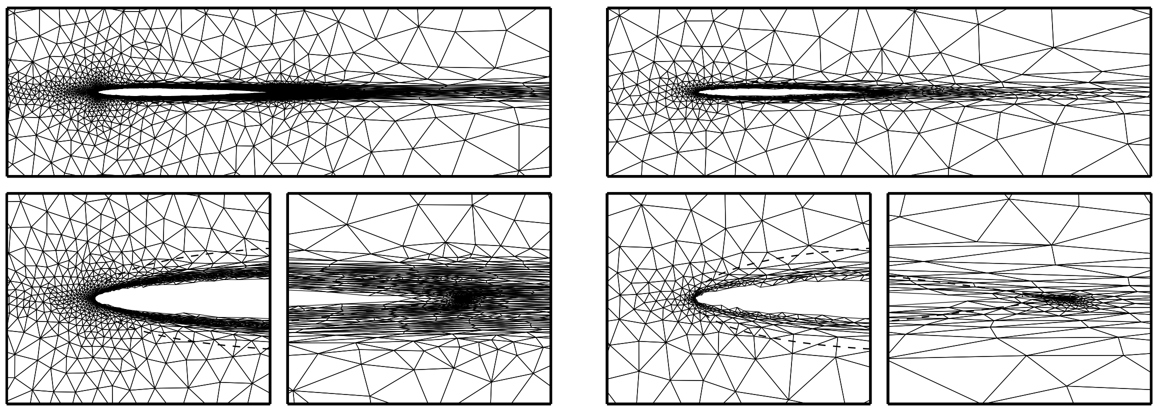


Figure 4-26: NACA 0005: $M_\infty = 0.4, Re = 50000, \alpha = 0^o$. Drag error versus degrees of freedom. Dashed line indicates prescribed tolerance of $e_0 = 0.01$ drag counts.

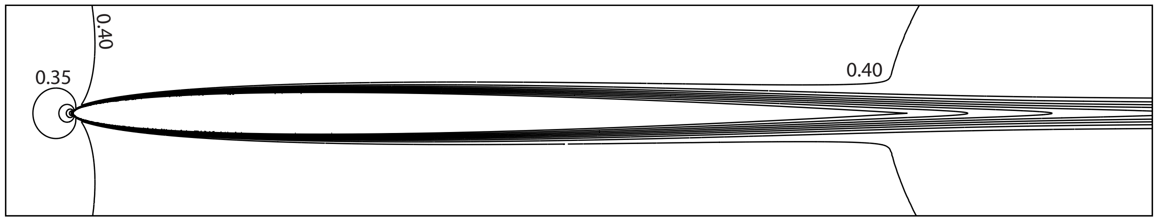
failure during the curving of the first layer of elements but does not hinder the cut-cell method.

Mach number contours for $p = 1$ and $p = 3$ on the final adapted meshes are shown in Figure 4-28. The $p = 1$ contours are very similar to the $p = 3$ contours. Compared to the $Re = 5000$ case, the boundary layer and wake are significantly thinner so that the Mach number anisotropy is more pronounced. The surface skin friction distributions on the adapted meshes are shown in Figure 4-29. C_f is nearly identical for the three orders.

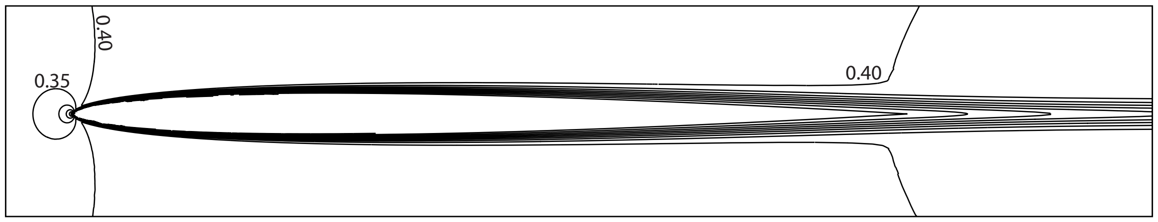


(a) $p = 2$: 5859 elements (b) $p = 3$: 2004 elements

Figure 4-27: NACA 0005: $M_\infty = 0.4, Re = 50000, \alpha = 0^o$. Final cut-cell meshes adapted on drag.



(a) Cut-cell, $p = 1$



(b) Cut-cell, $p = 3$

Figure 4-28: NACA 0005: $M_\infty = 0.4, Re = 50000, \alpha = 0^o$. Mach number contours for $p = 1$ and $p = 3$ on the final adapted cut-cell meshes.

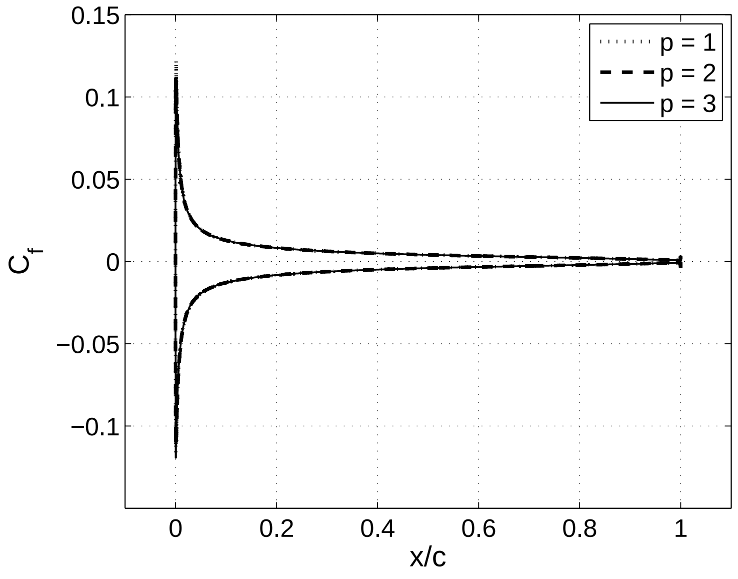


Figure 4-29: NACA 0005: $M_\infty = 0.4, Re = 50000, \alpha = 0^o$. Skin friction coefficient distributions on the final adapted meshes.

4.2.5 Joukowski Airfoil: High Pe Convection-Diffusion

One of the proposed advantages of the cut-cell method over the boundary-conforming method lies in the robustness of cut cells in dealing with anisotropic meshes near curved boundaries. This advantage has been demonstrated in the above results, notably in the $Re = 5000$, NACA 0012 runs and in the $Re = 50000$, NACA 0005 runs, in which the boundary-conforming method was prone to failure during curving of anisotropic boundary elements in post-processing. However, these cases arguably did not fully test the robustness of the cut-cell method. In terms of element aspect ratio, $Re = 50000$ is relatively benign compared to some of the practical high Reynolds numbers observed in cases of engineering interest. For example, a wing cross-section of a typical transport aircraft at cruise condition often faces Reynolds numbers in excess of 10^7. Furthermore, at such Reynolds numbers, the flow is turbulent, and accurate prediction of quantities such as skin friction and heat transfer demands resolution down to the viscous sublayer. In the presence of very high aspect ratio elements resulting from this required resolution, the cut-cell method faces a robustness challenge stemming from cutting a mesh whose edges in the boundary layer are nearly parallel to the geometry. In addition, a challenge for the adaptive algorithm is to robustly mesh the areas of high anisotropy.

An ideal test of the cut-cell adaptive method would be a Reynolds-averaged Navier-Stokes (RANS) simulation. However, as RANS discretization and solution is currently under development, a simpler equation set, convection-diffusion, was chosen to assess the robustness of the cut-cell adaptive method. The convection-diffusion equation is given by

$$\nabla \cdot (\mathbf{V}T) - \nabla \cdot (\nu \nabla T) = 0, \qquad Pe = \frac{V_\infty L}{\nu}, \tag{4.7}$$

where T is the scalar of interest, $\mathbf{V}$ is a prescribed velocity field, ν is the diffusion coefficient, V_∞ is a constant farfield velocity, L is a reference length scale, and Pe is the Peclet number, measuring the strength of convection relative to diffusion. The discontinuous Galerkin discretization of (4.7) proceeds similarly to that of the Navier-Stokes equations. Specifically, full upwinding is used for the convection term and the second form of Bassi & Rebay, described in Chapter 2, is used for the diffusion term. As neither $\mathbf{V}$ nor μ are assumed to be a function of T, (4.7) is a linear equation.

For the convection-diffusion equation, boundary-layer behavior can be observed at high Pe when a boundary condition specifies a T different from that in the bulk flow. To be concrete, in the following examples T will represent a temperature field, and the case of interest will be a heated airfoil in a high-speed flow. To allow for an analytical specification of the required velocity field in (4.7), potential flow around a Joukowski airfoil geometry is used. The velocity field is obtained via a conformal mapping of the velocity potential for non-lifting flow over a cylinder. Specifically, in a complex plane Z, non-lifting flow over a circular cylinder with center z_0 and radius R is governed by a complex potential, $f(z)$, that consists of a freestream flow superimposed with a doublet,

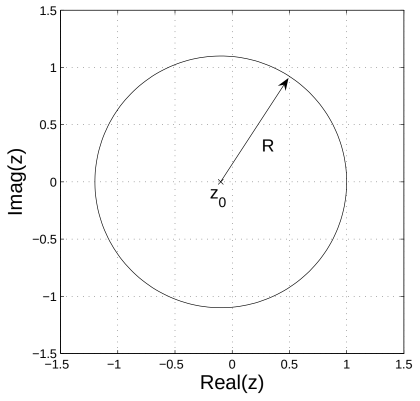
$$f(z) = V_\infty \left(z + \frac{R^2}{z - z_0} \right),$$

where V_∞ is the magnitude of the freestream velocity. The complex velocity at any point z is then given by $V_Z(z) = df/dz$. That is, the real component of V_Z is the "x-velocity", and the imaginary component is the "y-velocity". The cylinder in the Z plane is mapped to an airfoil in a new plane, W, via the conformal mapping,

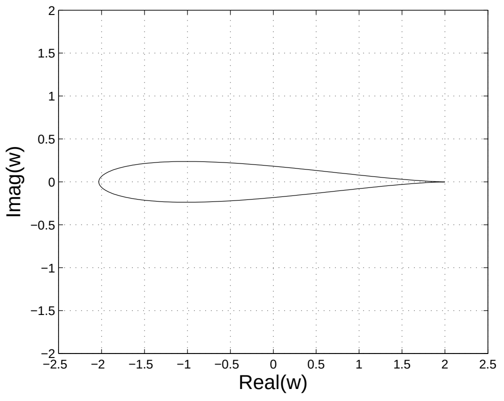
$$w = z + \frac{1}{z}.$$

A property of conformal mappings is that harmonic functions remain harmonic. Thus, since $f(z)$ is harmonic, so is $f(w)$, and hence the velocity over the Joukowski airfoil is given by $V_W(w) = df/dw = (df/dz)(dz/dw)$. In this work, a symmetric airfoil is considered, obtained from a cylinder with center $z_0 = (-0.1, 0)$ and radius $R = 1.1$, as shown in Figure 4-30.

The Peclet number, Pe, is chosen to simulate boundary-layer resolution required for a high Re, turbulent, Navier-Stokes flow. Simply setting $Pe = Re$ is not sufficient because a



(a) Cylinder in Z plane



(b) Airfoil in W plane

Figure 4-30: Joukowski transformation from a cylinder to an airfoil with a cusped trailing edge. The origin of the cylinder is on the real axis, which means that the resulting Joukowski airfoil is symmetric.

turbulent boundary layer contains an inner viscous sublayer that must also be resolved. A more realistic choice for Pe is one for which the thickness of the simulated scalar boundary layer matches the thickness of the turbulent inner layer. This inner layer extends to $y^+ \sim O(10)$, where y^+ is a non-dimensional wall distance, related to the physical wall distance, y, via $y^+ = (y/L)Re\sqrt{C_f/2}$. Specifically, in this work, $y^+ = 10$ is used as an estimate for the inner layer thickness. C_f, the skin friction coefficient, can be empirically correlated with Re. One such fit, due to Karman and Schoenherr, reads

$$\frac{1}{\sqrt{C_f}} \approx 4.15 \log_{10}(Re\ C_f) + 1.7. \tag{4.8}$$

More information on this fit, as well as turbulent boundary layers in general can be found in [77]. For a specified Re, (4.8) yields the corresponding C_f; y/L as a function of y^+ then follows from the definition of y^+.

As mentioned previously, $Re = 10^7$ is a representative value for an aircraft at cruise. At this Re and $y^+ = 10$, the above analysis yields $y/L \approx 2.8 \times 10^{-5}$. The Pe corresponding to an equivalent scalar boundary layer is found from a Blasius fit, $y_{99}/L \approx 5/\sqrt{Pe}$, where y_{99} is the 99% thickness [77]. Setting $y_{99}/L = 2.8 \times 10^{-5}$ yields $Pe \approx 4 \times 10^{10}$, over three orders of magnitude higher than the Re value.

The following sections present cut-cell, adaptive results for three Peclet numbers: $Pe = 4 \times 10^6$, $Pe = 4 \times 10^8$, and $Pe = 4 \times 10^{10}$. These conditions correspond to an equivalent $y^+ = 10$ inner-layer thickness for $Re = 7 \times 10^4$, $Re = 10^6$, and $Re = 10^7$, respectively. The freestream temperature is set to $T = 1$, while at the airfoil surface, a Dirichlet boundary condition of $T = 1.2$ is prescribed. The temperature difference is therefore $\Delta T = 0.2$. A velocity field with $V_\infty = 1$ is prescribed. The output of interest in each adaptive run is the total heat flux into the airfoil, given by

$$J = \sum_{\sigma^{bf} \in \Gamma_{\text{airfoil}}} \int_{\sigma^{bf}} \left[-\nu \partial_i T n_i + \eta^{bf} \delta_i^{bf} n_i \right] ds, \tag{4.9}$$

where Γ_{airfoil} is the airfoil surface consisting of boundary edges σ^{bf}, n_i are components of the normal vector pointing out of the computational domain, and δ_i^{bf} are components of the auxiliary variable associated with the BR2 discretization of the diffusion term. Summation is implied on the repeated index $i \in [1, .., 2]$. As in (4.6), the auxiliary variable term is necessary to ensure dual-consistency. The error tolerance in each case is set to 1% of an order of magnitude estimate for the output,

$$\bar{J} = \nu \Delta T \sqrt{Pe}.$$

This estimate is derived using $(\Delta T)/\delta$ to approximate the heat flux, where $\delta/L = 1/\sqrt{Pe}$ is an estimate of the distance over which heat transfer occurs. Finally, the geometry of the Joukowski airfoil is represented using a spline with 3200 knots. Such high accuracy in the geometry representation is required to ensure that the airfoil boundary remains parallel to the analytical velocity field.

$Pe = 4 \times 10^6$

The initial mesh used for the adaptive runs at $Pe = 4 \times 10^6$ is shown in Figure 4-31. It was obtained by adapting a uniform square mesh to the geometry. The adaptive method was run starting from this initial mesh for interpolation orders of $p = 1$, $p = 2$, and $p = 3$. The resulting output error convergence versus degrees of freedom is given in Figure 4-32a with the horizontal line indicating the error tolerance. The true value for the heat flux was taken from a $p = 3$ solution on a uniformly-refined finest-level $p = 2$ mesh. While both $p = 2$ and $p = 3$ achieve the desired heat flux error tolerance with slightly over 10000 DOF, $p = 1$ requires over 40000 DOF. Thus, in terms of estimated work, $p = 2$ is the least expensive by a factor of 2 compared to $p = 1$. Another useful measure for this case is the aspect ratio, AR, of the elements in the final meshes. A histogram of these aspect ratios is given in Figure 4-32b. This histogram was generated using 20 bins, distributed logarithmically between 0 and 10^4. The count in each bin was normalized by $20/N_{\text{elem}}$ to account for differing number of elements in the meshes. While the average AR for $p = 1$ is 36, it is 15 for $p = 2$ and 10 for $p = 3$.

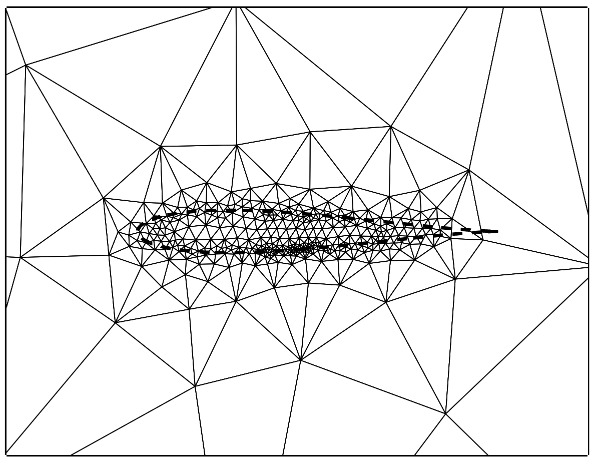
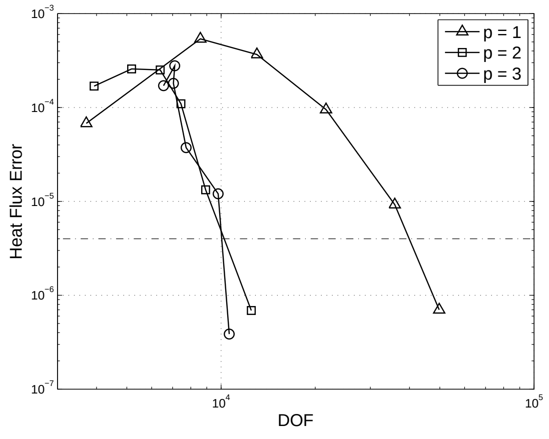
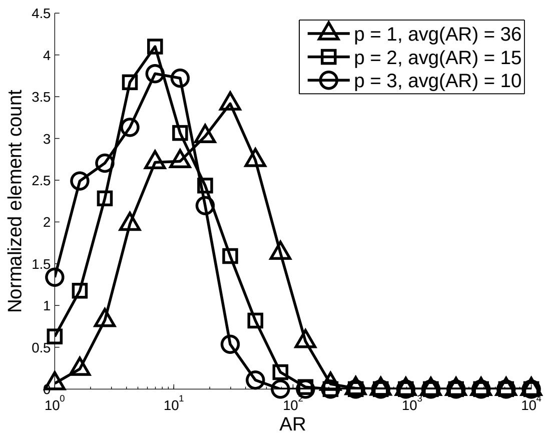


Figure 4-31: Initial mesh for $Pe = 4\times10^6$ computation, adapted to geometry: 904 elements. Also shown is the airfoil spline representation.



(a) Adaptation history



(b) AR histogram

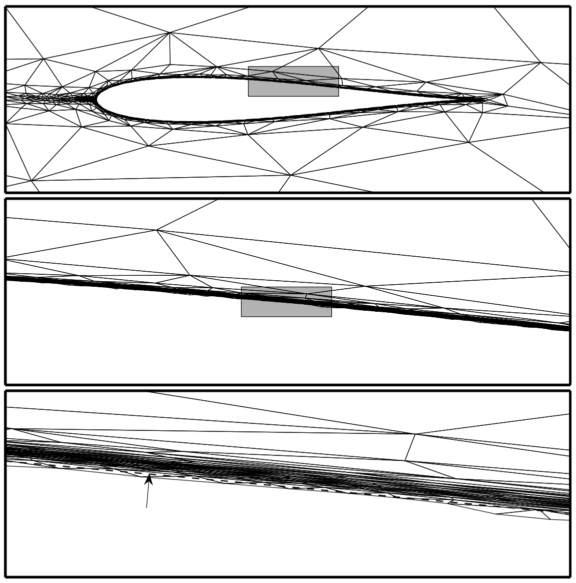
Figure 4-32: Joukowski airfoil: $Pe = 4 \times 10^6$. Output error versus DOF adaptation history and a histogram of element aspect ratio in the final adapted meshes.

The final adapted meshes for $p = 1, 2, 3$ are shown in Figure 4-33. The wake is not resolved because it does not significantly affect the heat transfer at the airfoil. A sequence of magnified images is shown for each mesh to illustrate the relative thickness of the boundary layer. Triangles of the background mesh that are completely contained within the airfoil geometry are not shown, giving the illusion of boundary conforming meshes due to the geometry-aligned anisotropic elements in the boundary layer. The coarseness in the boundary layer mesh allowed by $p > 1$ is clearly evident. A rough count of the average number of cells within the boundary layer in a direction normal to the airfoil boundary yields about 25-30 for the final adapted $p = 1$ mesh, 5-6 for the final adapted $p = 2$ mesh, and 2-3 for the final adapted $p = 3$ mesh. The coarser meshes for higher p not only result in fewer degrees of freedom but also take some of the burden off the mesh generator. This fact becomes particularly important at the higher Peclet numbers.

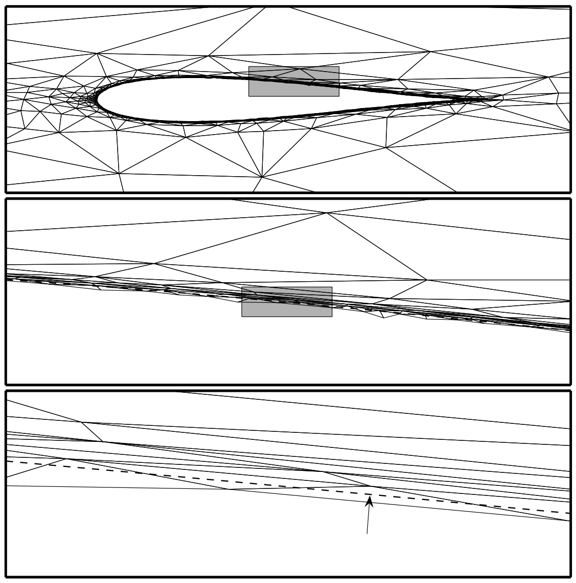
Finally, Figure 4-34 shows the heat transfer coefficient, C_H, for the solutions on the final adapted $p = 1, 2, 3$ meshes. This coefficient is computed via

$$C_H = \frac{-\nu \partial_i T n_i + \eta^{bf} \delta_i^{bf} n_i}{V_\infty \Delta T},$$

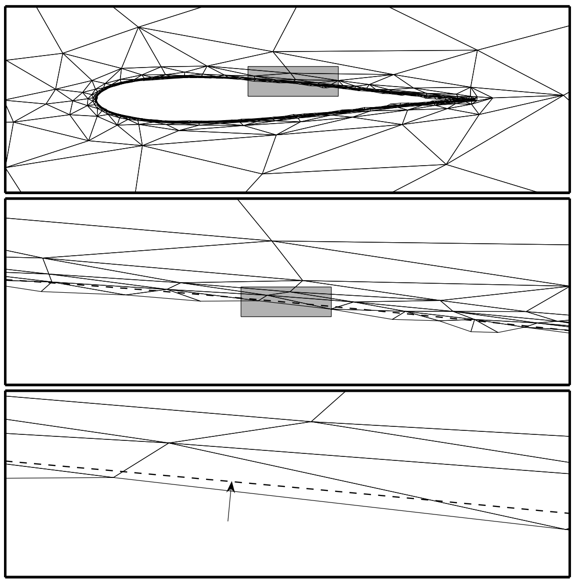
where the δ_i^{bf} term is included for dual-consistentcy, as in (4.9). ΔT is the difference between the airfoil surface temperature and the freestream temperature. C_H exhibits oscillatory behavior, especially near the leading edge.



(a) $p = 1$

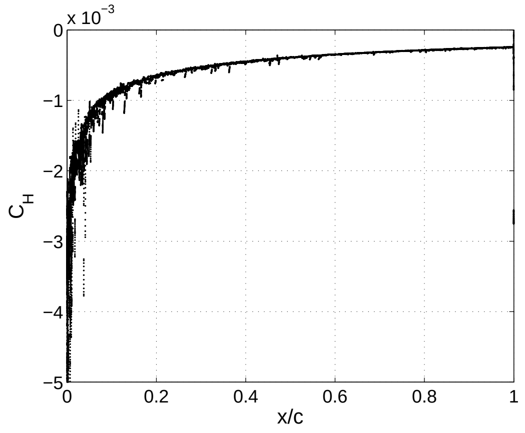


(b) $p = 2$

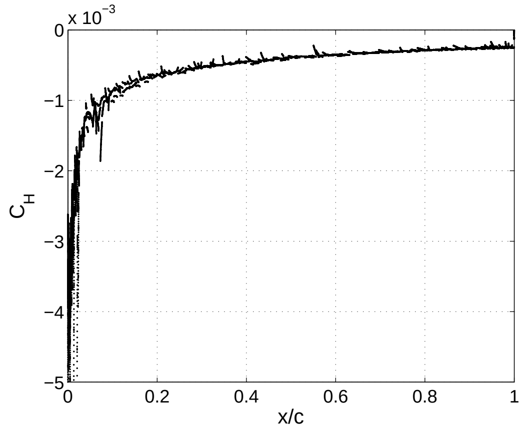


(c) $p = 3$

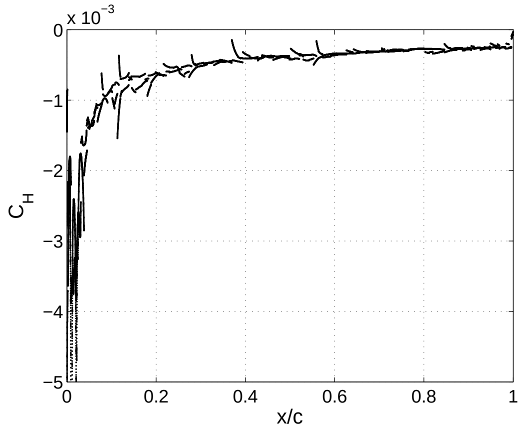
Figure 4-33: Joukowski airfoil: $Pe = 4 \times 10^6$. Final adapted meshes for $p = 1$, $p = 2$, and $p = 3$. Shaded areas indicate zoom regions for the subsequent plots. Arrows point to the dashed line marking the embedded airfoil boundary.



(a) $p = 1$



(b) $p = 2$



(c) $p = 3$

Figure 4-34: Joukowski airfoil: $Pe = 4 \times 10^6$. Heat transfer coefficient along the airfoil surface on the final adapted $p = 1, 2, 3$ meshes. Negative heat transfer corresponds to heat flux into the flow, out of the airfoil.

The refined meshes in Figure 4-33 exhibit very little refinement of the wake. However, the output error indicator is not a bound, but rather an estimate that improves with resolution. Thus, the fact that the wake is not refined may be due to poor wake resolution in the initial mesh. To verify whether wake refinement is truly not important for the heat flux calculation in this case, adaptation was also performed starting from an initial mesh with significant wake resolution. This initial mesh is shown at the top of Figure 4-35. It was generated by adapting to a temperature line integral output along the wake. Starting with this mesh, the adaptive method was run for $p = 1$ using heat flux at the airfoil as the output

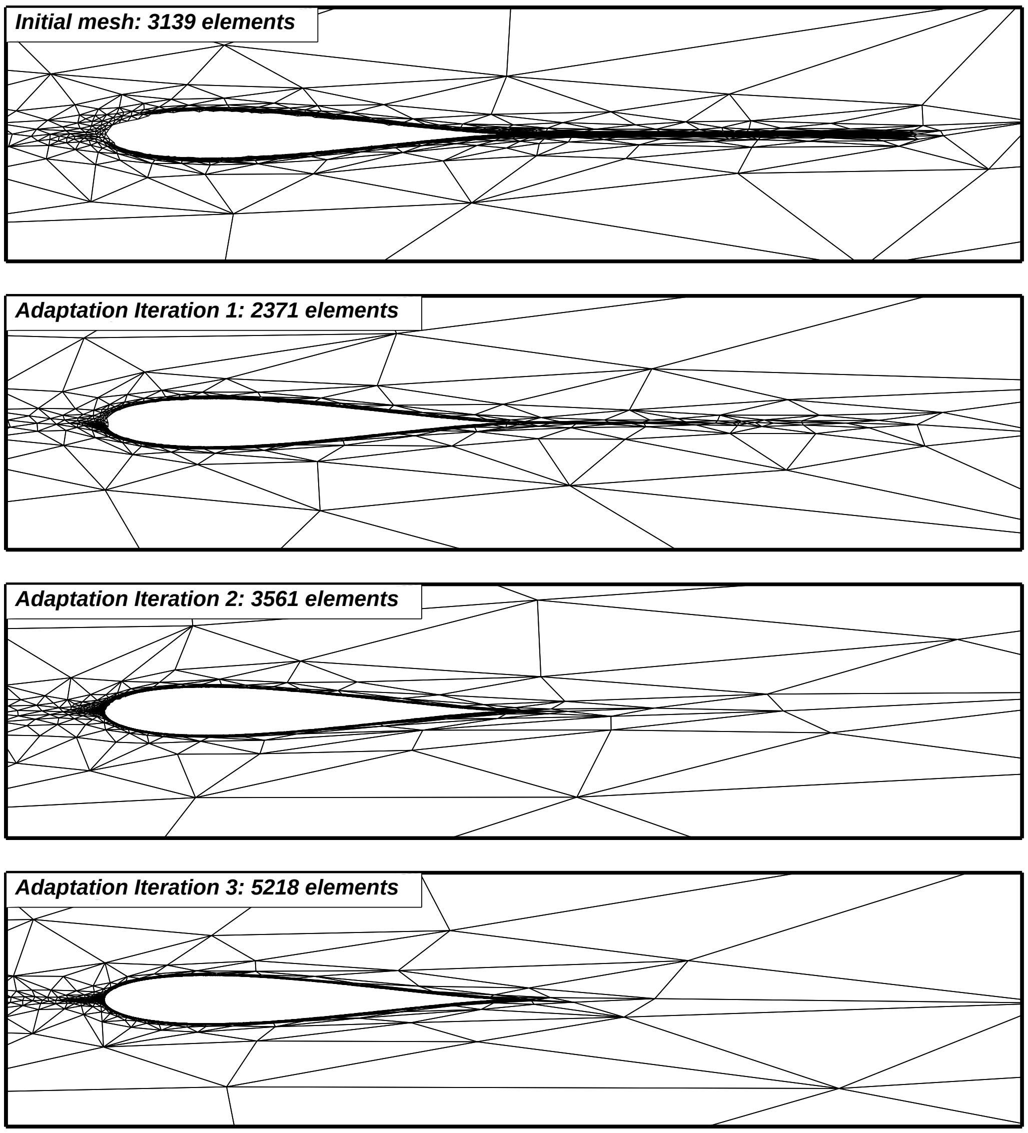

Initial mesh: 3139 elements
Adaptation Iteration 1: 2371 elements
Adaptation Iteration 2: 3561 elements
Adaptation Iteration 3: 5218 elements

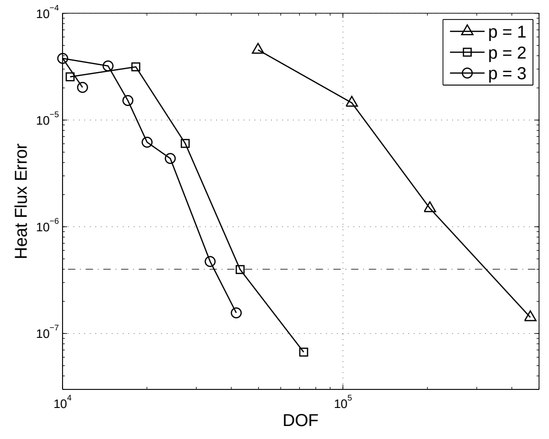
Figure 4-35: Joukowski airfoil: $Pe = 4 \times 10^6$. $p = 1$ adaptation on heat flux starting from an initial mesh with significant wake resolution. The wake is coarsened at each adaptation iteration.

of interest. The meshes from the first three adaptation iterations are shown in Figure 4-35. Clearly, the mesh in the wake is significantly coarsened at each adaptation iteration. Thus, wake resolution does not affect the airfoil heat flux in this case, an observation that can be explained by the fact that the magnitude of diffusion is very small compared to convection and that there is no coupling between temperature and velocity.

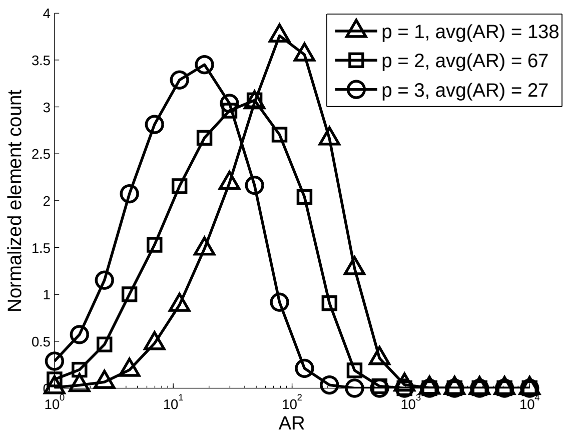
$Pe = 4 \times 10^8$

For the $Pe = 4 \times 10^8$ case, the estimate of the output error calculated from a solution on the initial mesh in Figure 4-31 is below the error tolerance ($.01\bar{J}$), even though the true output error is not. As discussed in Section 3.1, although the output error estimate converges to the true error as the solution becomes more accurate, it does not constitute a bound for the error. To remedy this situation, the error tolerance can be reduced, at least for the first few iterations, or a finer initial mesh can be used. In order to keep the error tolerances consistent, the latter approach is taken in this work so that the final adapted meshes from the $Pe = 4 \times 10^6$ runs at each order serve as the starting meshes.

The adaptive method was run at the new Pe and e_0 for interpolation orders of $p = 1, 2, 3$. The resulting output error convergence histories versus degrees of freedom are given in Figure 4-36a. As in the previous section, a $p = 3$ solution on a uniformly-refined final adapted $p = 2$ mesh was used to calculate the true value of the heat flux. The difference in DOF at error tolerance between $p = 1$ and $p = 2, 3$ is now greater: $p = 1$ requires almost an order of magnitude more degrees of freedom, which corresponds to about a factor of 5 increase in work compared to $p = 2$. $p = 3$ performs slightly better than $p = 2$ in terms of DOF, but slightly worse in terms of estimated work. Relative to the $Pe = 4 \times 10^6$ case, the increases in degrees of freedom for $Pe = 4 \times 10^8$ are $10, 5, 4$ for $p = 1, 2, 3$, respectively. Thus, the relative DOF increase for $p = 2$ and $p = 3$ is lower than for $p = 1$.



(a) Adaptation history



(b) AR histogram

Figure 4-36: Joukowski airfoil: $Pe = 4 \times 10^8$. Output error versus DOF adaptation history and a histogram of element aspect ratio in the final adapted meshes.

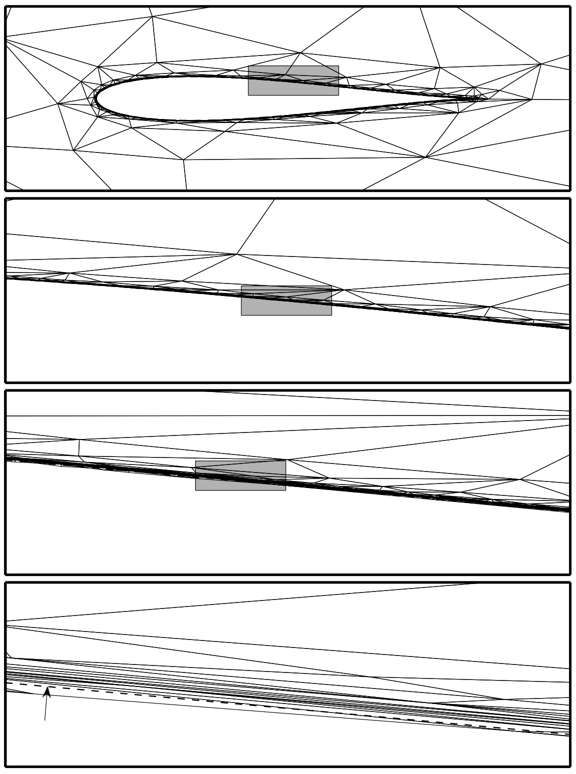
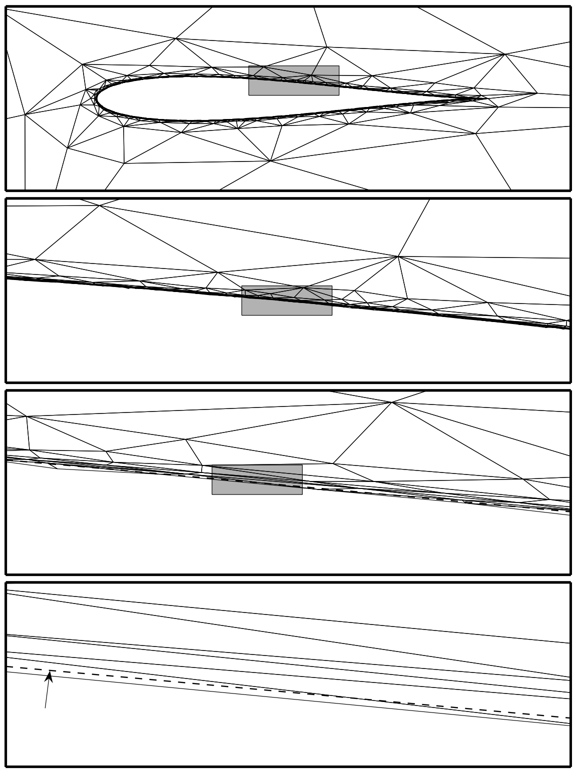
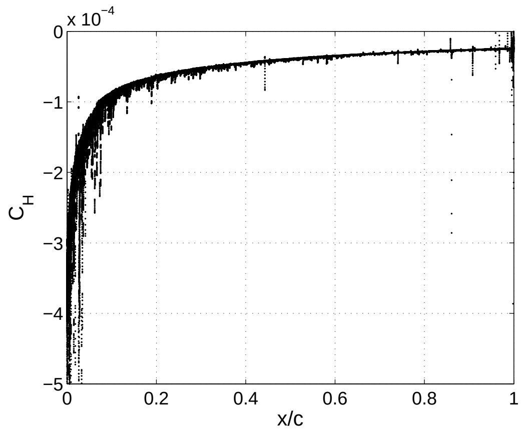
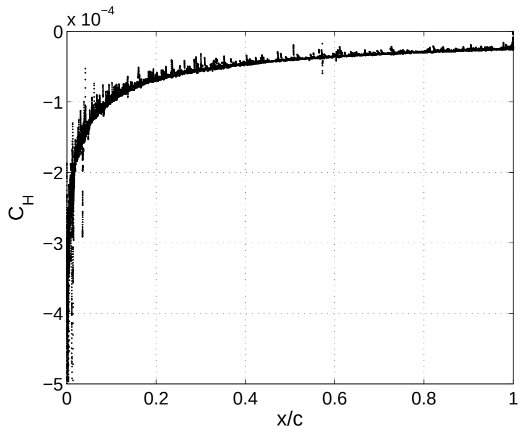


(a) $p = 2$ (b) $p = 3$

Figure 4-37: Joukowski airfoil: $Pe = 4 \times 10^8$. Final adapted meshes for $p = 2$ and $p = 3$. Shaded areas indicate zoom regions for the subsequent plots. Arrows point to the dashed line marking the embedded airfoil boundary.

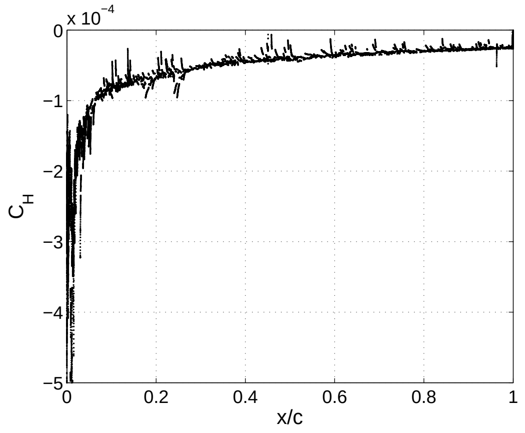
A histogram of the element aspect ratios in each of the final adapted meshes is shown in Figure 4-36b. Again, $p = 3$ exhibits the lowest average aspect ratio of 27, while $p = 1$ exhibits the highest of 138. Figure 4-37 shows the final adapted meshes for $p = 2$ and $p = 3$. Again, a sequence of magnified meshes is used to depict the thickness of the boundary layer relative to the geometry. The final $p = 1$ mesh is not shown as, at the same magnification, the individual elements are not discernible. An approximate count of the average number of cells within the boundary layer in the normal direction yields about 35-40 for the final adapted $p = 1$ mesh, 5-6 for the final adapted $p = 2$ mesh, and 3 for the final adapted $p = 3$ mesh. These numbers are similar to the $Pe = 4 \times 10^6$ case with the exception that the $p = 1$ count is slightly higher. Figure 4-38 shows the heat transfer coefficient, C_H, for the solutions on the final adapted $p = 1, 2, 3$ meshes. As in the $Pe = 4 \times 10^6$ case, C_H exhibits oscillatory behavior, especially near the leading edge.



(a) $p = 1$



(b) $p = 2$



(c) $p = 3$

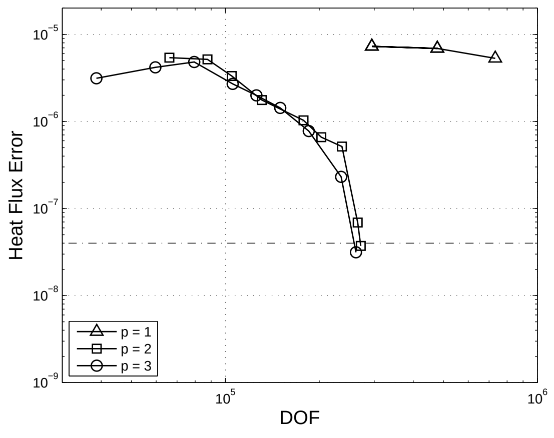
Figure 4-38: Joukowski airfoil: $Pe = 4 \times 10^8$. Heat transfer coefficient along the airfoil surface on the final adapted $p = 1, 2, 3$ meshes.

$Pe = 4 \times 10^{10}$

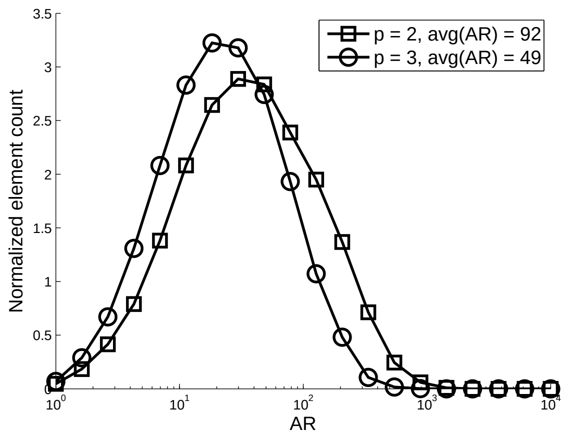
As discussed at the beginning of this section, $Pe = 4 \times 10^{10}$ results in a boundary layer that is of comparable thickness to a turbulent inner layer in a typical cruise-configuration simulation. As in the $Pe = 4 \times 10^8$ case, a solution on a coarse initial mesh yields an error estimate below the tolerance $(0.01\bar{J})$. Thus, the initial meshes for the adaptive runs consist of the final adapted meshes from the $Pe = 4 \times 10^8$ runs.

The adaptive, output-error convergence results for $p = 1, 2, 3$ are shown in Figure 4-39a. As in the previous cases, a $p = 3$ solution on a uniformly-refined adapted $p = 2$ mesh served as the true solution. Only three adaptation iterations are shown for $p = 1$ because the mesher failed to return a valid mesh after the third adaptation iteration. Specifically, the returned mesh contained triangles with areas that were negative and close to machine precision. This failure is likely due to areas of very high anisotropy requested in the $p = 1$ adaptive run.

On the other hand, both $p = 2$ and $p = 3$ converged successfully to satisfy the error tolerance. The convergence histories between the two are very similar. For both $p = 2$ and $p = 3$, the degree of freedom count at the error tolerance is roughly 6 times greater than for the $Pe = 4 \times 10^8$ case. A histogram of the element aspect ratios in the final $p = 2$ and $p = 3$ adapted meshes is shown in Figure 4-39b. The average element aspect ratio is 92 for $p = 2$ and 49 for $p = 3$.



(a) Adaptation history



(b) AR histogram

Figure 4-39: Joukowski airfoil: $Pe = 4 \times 10^{10}$. Output error versus DOF adaptation history and a histogram of element aspect ratio in the final adapted meshes.

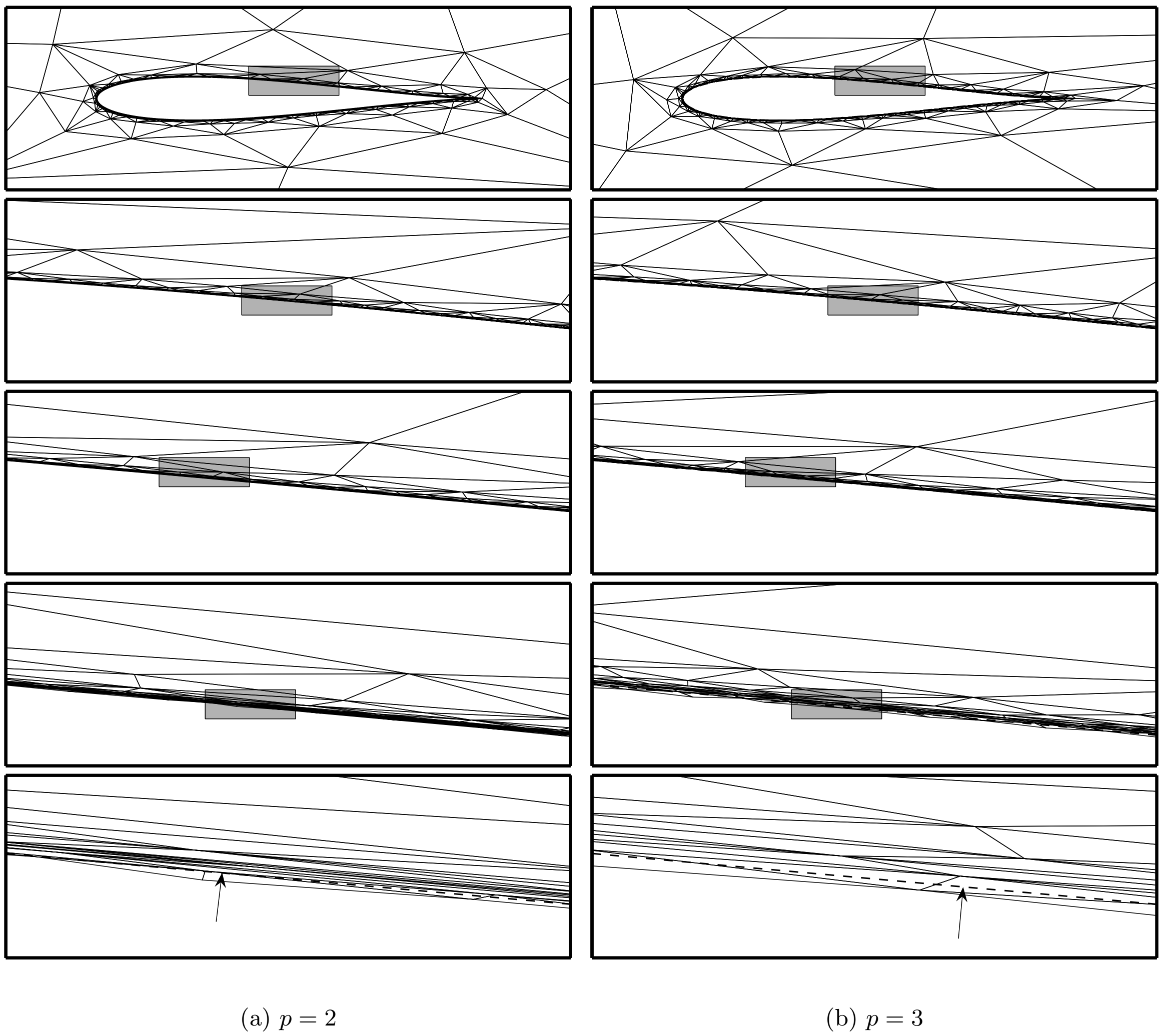
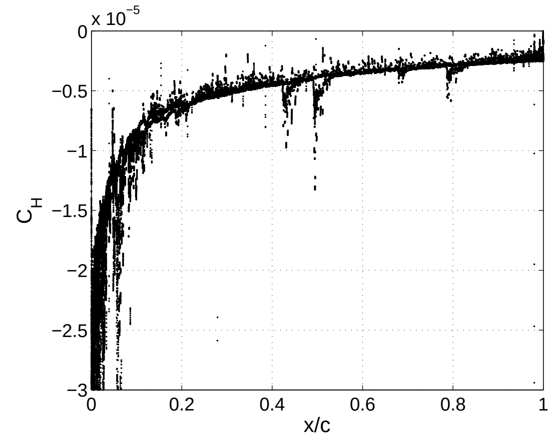
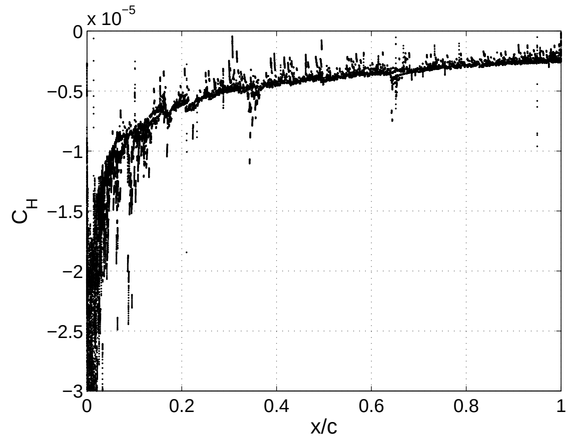

(a) $p = 2$ (b) $p = 3$

Figure 4-40: Joukowski airfoil: $Pe = 4 \times 10^{10}$. Final adapted meshes for $p = 2$ and $p = 3$. Shaded areas indicate zoom regions for the subsequent plots. Arrows point to the dashed line marking the embedded airfoil boundary.

The final adapted meshes for $p = 2$ and $p = 3$ are shown in Figure 4-40. Note the four levels of magnification compared to the two for $Pe = 4 \times 10^6$ and three for $Pe = 4 \times 10^8$. Away from the boundary layer, the meshes are nearly identical. Inside the boundary layer, $p = 3$ again admits a coarser mesh with an average of 3 elements within the boundary layer, compared to 5-6 for $p = 2$. The fact that these meshes, with edges nearly parallel to the geometry, were successfully cut demonstrates the robustness of the cut-cell method for modeling boundary layers in practical simulations. Finally, Figure 4-41 shows the heat transfer coefficient, C_H, for the solutions on the final adapted $p = 2$ and $p = 3$ meshes. Again, C_H exhibits oscillatory behavior, especially near the leading edge.



(a) $p = 2$



(b) $p = 3$

Figure 4-41: Joukowski airfoil: $Pe = 4 \times 10^{10}$. Heat transfer coefficient along the airfoil surface on the final adapted $p = 2$ and $p = 3$ meshes.

Chapter 5

Cut Cells in Three Dimensions

The two-dimensional cut-cell method presented in Chapter 4 demonstrates the feasibility of using simplex cut cells in practical aerodynamic computations. However, the problem of robust, boundary-conforming mesh generation in two dimensions is not insurmountable. Current meshers can handle complex geometries and curved boundaries can often be dealt with in a systematic manner. Nevertheless, the two-dimensional cut-cell method provides a relatively quick and easy start-up alternative to boundary conforming meshes. The only requirement is a mesher that can mesh a simple background area such as a rectangle, according to a prescribed metric. Geometry information is never passed to the mesher, eliminating the need for a common geometry format or the loss of geometry information. Furthermore, robustness of the two-dimensional cut-cell method has been demonstrated even for difficult problems, such as high-Peclet number convection-diffusion flow.

While two-dimensions offer several difficult problems suitable for cut cells, the potential impact of cut cells in three dimensions is much greater. As mentioned in Chapter 1, there is currently no robust three-dimensional boundary-conforming mesher that can handle curved boundaries for general, anisotropic meshes. A three-dimensional cut-cell method that produces results similar to the two-dimensional cut-cell method is quite desirable. Therefore, this chapter explores the feasibility of a practical three-dimensional cut-cell implementation. The challenges in three dimensions include a well-conditioned high-order integration technique, a suitable geometry definition, and a robust cutting algorithm. The following sections explore one possible set of solutions to these challenges.

5.1 Cutting and Integration Mechanics

5.1.1 Geometry Definition

Three-dimensional surface modeling is a broad field, with many geometry representation techniques. Computer-Aided Design packages typically employ one or more of a variety of spline representations, including bivariate splines and non-uniform rational B-splines (NURBS) [21]. For CFD purposes, the surface representation should be watertight, which means that no gaps should be present at surface junctures. While general CAD models are not always watertight, robust post-processing tools are available for generating watertight descriptions. One of these tools is the Computational Analysis Programing Interface (CAPRI) [33], which supports a wide variety of CAD models. CAPRI is used in this work, and the resulting watertight description is treated as the exact geometry for the CFD calculations. An ideal cut-cell method would use the CAD or exact geometry to perform the intersections with the background mesh and to obtain the integration rules. However, supporting general CAD models requires ability to perform intersections with a variety of surface representations, including NURBS and bivariate splines. Such capability is beyond the scope of this initial proof-of-concept demonstration.

An alternate approach to supporting CAD directly is to use an intermediate surface representation for which the cut-cell implementation is simplified. The desired qualities of such an intermediate representation are as follows:

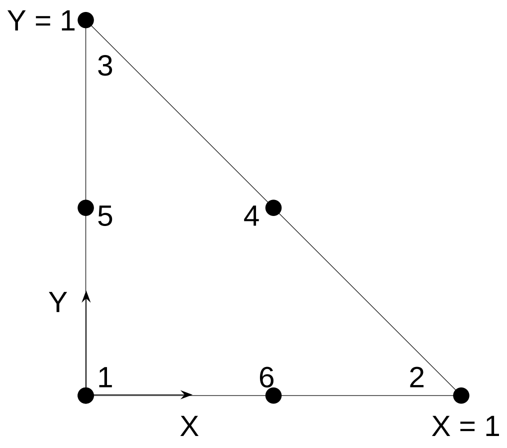
- Watertight: no gaps in the surface.
- Easy to construct from existing CAD representations.
- Possible to intersect analytically with simplex elements (tetrahedra).
- Suitable for high-order finite element computations.

One common representation used in CFD consists of surface tessellations with linear triangular patches. Tolerances on deviation from the exact geometry necessitate high refinement in areas where the surface curvature is large. These linear patch representations usually serve as the first step in a boundary-conforming volume mesh generation process, although recently they are also being used in finite volume cut-cell methods [56]. Linear patches can represent a surface in a watertight fashion and can be intersected analytically with linear elements via plane-plane intersections. In addition, tools are available for constructing surface tessellations from CAD models. For example, CAPRI provides this capability and also ensures that the obtained linear tessellation is watertight.

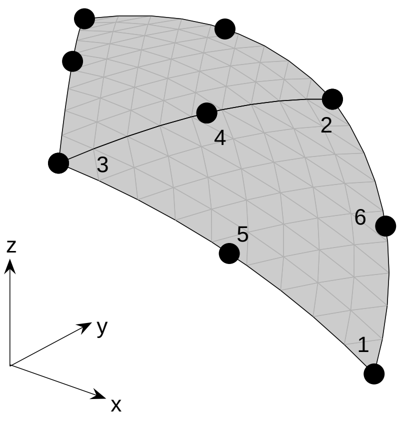
A drawback of using linear triangular patches for geometry representation is that they are not very well-suited for high-order finite element computations. The problem lies in the relatively large geometry slope discontinuities that can occur between linear patches. A sufficiently high-order flow discretization will resolve these discontinuities, leading to singularities in the solution. Physically, such singularities are expected because flow around an infinitely-sharp corner requires a point of infinite acceleration, which produces a pressure spike. Bassi and Rebay demonstrate this problem for a two-dimensional boundary-conforming DG method [5]. Specifically, they find that a linear geometry representation with $p > 0$ produces a non-physical entropy wake for flow around a cylinder. This wake disappears for higher-order geometry representations, in which the slope discontinuities are greatly diminished.

For cut-cell computations using linear patches, the magnitudes of the geometry slope discontinuities can be controlled independently of the computational mesh. Specifically, slope discontinuities can be diminished by refining the surface triangulation. From experience, when the magnitude of the slope discontinuities drops below a certain level relative to the resolution of the computational mesh, the spurious corner singularities are no longer resolved. For example, a standard boundary-conforming technique for curving boundary elements relies on isoparametric or superparametric elements in which additional geometry nodes are placed on the exact geometry but in which slope continuity is not necessarily enforced between neighboring boundary elements. As shown by Bassi and Rebay [5], the solutions obtained on such meshes, which effectively contain slope discontinuities, do not possess spurious features and exhibit optimal accuracy convergence. Therefore the effect of a slope discontinuity on the solution depends on the computational mesh resolution, i.e. mesh refinement and interpolation order, in the vicinity of the discontinuity.

Experiments in 2D have shown that using linear patches for high-order computations requires a prohibitively large number of patches to sufficiently resolve the surface in terms of maximum allowed slope discontinuity. As demonstrated in Appendix F, this effect is due to the first-order slope error convergence for linear patches: increasing the number of linear patches with refinement only linearly affects the slope discontinuities. To alleviate this problem, an alternate intermediate representation is proposed in this work: quadratic patches. Quadratic patches differ from linear patches by adding extra nodes on the geometry at the midpoints of the surface triangle edges. Quadratic interpolation is then used to represent the surface on each patch. An example of two adjacent quadratic patches is shown in Figure 5-1b. Also illustrated in the figure is a local node numbering for one of the patches. The surface representation on each quadratic patch can be written as a mapping



(a) Reference triangle



(b) Two adjacent patches

Figure 5-1: Patch reference triangle (a) and an example of two adjacent patches (b). The first three nodes in the numbering are the "linear nodes", whereas the latter three are the "high-order nodes."

from a unit reference triangle, shown in Figure 5-1a, via

$$\mathbf{x} = \sum_{j=1}^{6} \phi_j(\mathbf{X})\mathbf{x}_j. \tag{5.1}$$

In this equation, $\mathbf{x} = [x, y, z]^T$ is a vector of the three-dimensional coordinates of the patch in physical space, $\mathbf{X} = [X, Y]^T$ is a vector of the two-dimensional coordinates in reference space, and the $\mathbf{x}_j$ vectors are physical-space coordinates of the six patch nodes. The convention for the ordering in physical space is such that the vector obtained via the right-hand rule in traversing nodes 1,2,3 (i.e. $\vec{12} \times \vec{13}$) points into the computational domain. In practice, the six nodes are placed on the exact geometry. The $\phi_j(\mathbf{X})$ are quadratic Lagrange interpolating functions in reference space, with nodes as shown in Figure 5-1a. For completeness, they are:

$$\begin{aligned}
\phi_1 &= 1 - 3X - 3Y + 2X^2 + 4XY + 2Y^2, & \phi_2 &= -X + 2X^2, \\
\phi_3 &= -Y + 2Y^2, & \phi_4 &= 4XY, \\
\phi_5 &= 4Y - 4XY - 4Y^2, & \phi_6 &= 4X - 4X^2 - 4XY.
\end{aligned}$$

Letting $\mathbf{R} = [X, Y, 1]^T$, each interpolating function can be expressed compactly in matrix form as $\phi_j = \mathbf{R}^T\mathbf{P}_j\mathbf{R}$, where the $\mathbf{P}_j$ are 3×3, symmetric coefficient matrices obtained from the above equations. For example, $\phi_1(\mathbf{X})$ can be written as

$$\phi_1(\mathbf{X}) = \underbrace{\begin{bmatrix} X & Y & 1 \end{bmatrix}}_{\mathbf{R}^T} \underbrace{\begin{bmatrix} 2 & 2 & -3/2 \\ 2 & 2 & -3/2 \\ -3/2 & -3/2 & 1 \end{bmatrix}}_{\mathbf{P}_0} \underbrace{\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}}_{\mathbf{R}}.$$

Quadratic patches are watertight because the interpolation of an edge depends only on the locations of the three nodes on that edge, so that the surface interpolations from two adjacent patches always match. An example of a patch representation of a portion of a sphere is shown in Figure 5-2. While the nodes of the patches lie exactly on the surface of the sphere, the quadratically-interpolated coordinates on the patch interiors and edges do not. An illustration of the interpolation error on a typical quadratic patch is provided in Appendix F. Thus, quadratic patches serve only as an approximation to the true geometry. However, compared to linear patches, quadratic patches are much more efficient

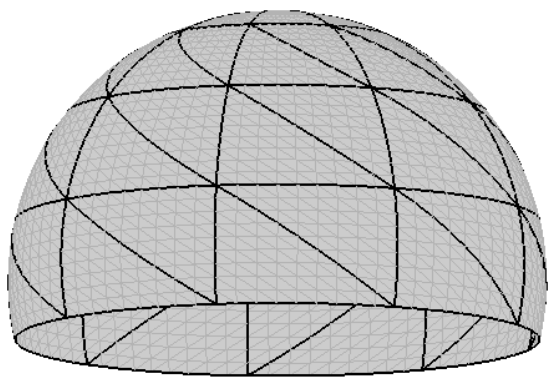


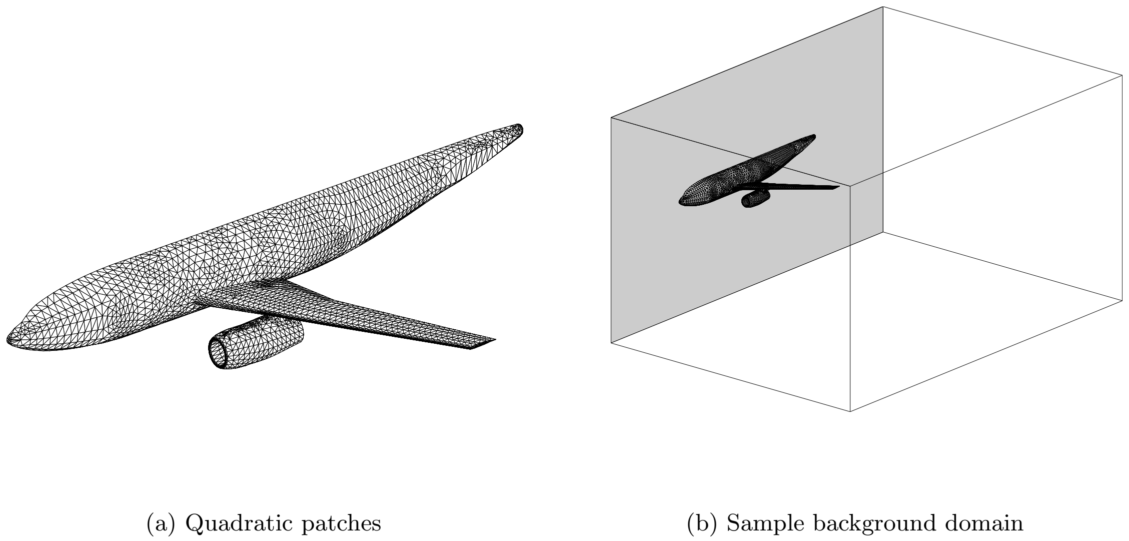
Figure 5-2: Example of a quadratic-patch representation of a portion of a sphere.

at minimizing the magnitudes of surface slope discontinuities. This point is illustrated in both two and three dimensions in Appendix F. The increased efficiency of quadratic patches means that many fewer surface patches are required for an adequate geometry representation.

Quadratic patches are not necessarily the ideal solution. Slope discontinuities are still present, which means that an adaptive method will likely require surface re-tessellation in areas where the volume mesh becomes highly-resolved. A more accurate or even exact geometry representation with tractable cut-cell intersection and integration algorithms would likely perform better and is an area of possible future work. Rather than seeking the most accurate technique from the start, the goal of this work is to demonstrate that a cut-cell method based on quadratic patches is possible and that it performs well for moderate orders of interpolation. The following sections describe in detail the mechanics of such a method.

5.1.2 Cutting Algorithm

The three-dimensional cutting algorithm takes as input a quadratic-patch surface representation of the geometry of interest and a linear, tetrahedral volume mesh of the background domain. As in two dimensions, the background domain consists of the computational domain and the interior of the geometry. An example of a quadratic-patch surface representation of a wing-body-nacelle geometry is shown in Figure 5-3a. Due to symmetry, only half of the geometry is modeled. Shown in Figure 5-3b is one possible choice for the background domain. In this case, it is a box; on five sides of the box, farfield boundary conditions are imposed, and the remaining side is a symmetry plane. For improved robustness of the cutting algorithm near the symmetry plane-body intersection, more than half of the geometry is provided. That is, the geometry extends a small distance past the symmetry plane, out of the background domain to prevent degeneracies in cutting. The flow solution is not affected by this extension because the cutting algorithm discards geometry outside the background domain.



(a) Quadratic patches

(b) Sample background domain

Figure 5-3: Quadratic patch surface representation of a wing-body-nacelle geometry (a) and one possible choice for the background domain (b). The shaded side of the background domain indicates a symmetry boundary condition. Farfield boundary conditions are applied on the other sides.

The output of the cutting algorithm is a cut-cell mesh of the computational domain obtained from the original background mesh by removing elements completely contained in the geometry and by appropriately cutting elements that intersect the geometry. The resulting cut cells are portions of the original tetrahedra that lie inside the computational domain. For example, Figure 5-4a shows an intersection between a background-mesh tetrahedron and a quadratic patch surface. The upper portion of the tetrahedron lies inside the computational domain and forms a cut cell. Ultimately, for use in the solver, integration rules are required on the interior and on the 2D boundary of such a cut cell. However, generating these rules first requires identification and description of the intersections that produce the cut cells.

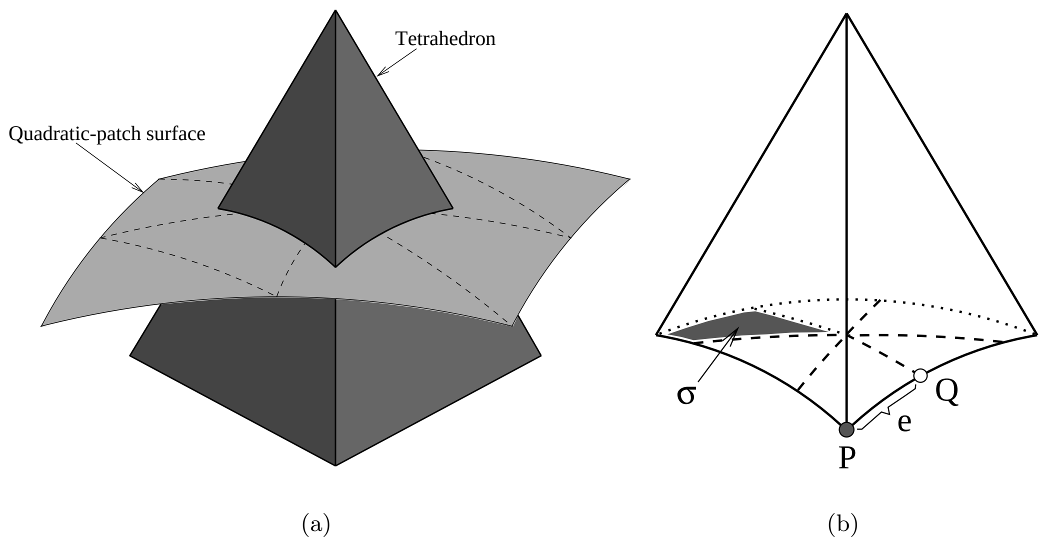


Figure 5-4: A background-mesh tetrahedron intersecting a quadratic-patch surface (a). The upper portion of the tetrahedron lies inside the computational domain. A wire-frame of the resulting cut-cell (b) is obtained by joining various intersection points (e.g. P and Q) into 1D structures (e.g. e). Loops of 1D structures enclose 2D structures, such as the shaded one labeled by σ.

Figure 5-4b illustrates the basic intersection features for the cut cell at hand. As shown, the cut cell is enclosed in a wire-frame of edges, or "1D structures." Each 1D structure is a possibly-curved line segment joining two intersection points. These intersection points consist of tetrahedron vertices lying inside the computational domain, intersections between tetrahedron edges and patches (e.g. point P in the figure), and intersections between patch edges and tetrahedron faces (e.g. point Q in the figure). The 1D structure labeled e connects points P and Q. Loops of 1D structures enclose "2D structures," which effectively become faces of the new cut cell. These "2D structures" consist of portions of the patches lying inside the tetrahedron as well as portions of the tetrahedron faces lying inside the computational domain.

An algorithm for constructing cut-cells must identify and connect the intersection points, 1D structures, and 2D structures. Before presenting the algorithm developed in this work, a key component is first described in detail: that of an intersection between a tetrahedron face and a quadratic patch.

Face-Patch Intersections as Conic Sections

A key component of the cutting algorithm is the intersection between a linear tetrahedron face and a quadratic patch. Figure 5-5 shows an example of one quadratic patch intersecting a tetrahedron. In this case, the quadratic patch intersects the interior of the tetrahedron and protrudes out of faces 1, 3, and 4 of the tetrahedron. Note, the local number of a tetrahedron face is the local number of the vertex not on that face. In the figure, the portion of the patch inside the tetrahedron is hidden from view. Intersections between the patch and the faces of the tetrahedron produce 1D structures required in describing the cut cell. Because the patch coordinates vary quadratically, the patch surface is a highly-nonlinear function of the physical coordinates. As such, intersections are difficult to perform in physical space. A more tractable approach is to work in the reference space of the quadratic patch, in which analytical intersections are possible.

Consider a single face of a tetrahedron. The coordinates of a point, $\mathbf{x}$, lying inside the tetrahedron or on its surface must satisfy

$$(\mathbf{x} - \mathbf{q}_f) \cdot \mathbf{n}_f \leq 0, \tag{5.2}$$

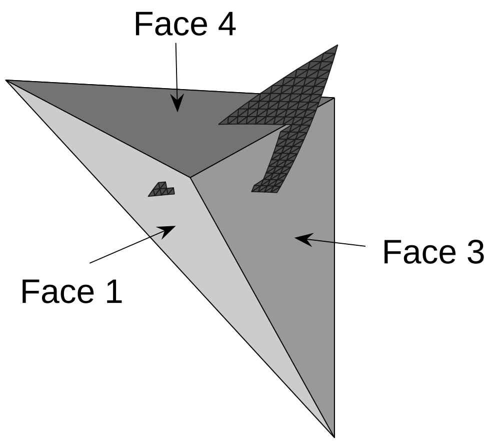


Figure 5-5: Intersection of a quadratic patch with a tetrahedron.

where $\mathbf{q}_f$ is an arbitrary point on the tetrahedron face, in practice taken to be one of the nodes on the face, and $\mathbf{n}_f$ is the outward-pointing normal associated with the face. A point, $\mathbf{x}$, satisfying this condition for all four faces necessarily lies inside the tetrahedron or on its surface. If the inequality is not satisfied for one or more faces, then $\mathbf{x}$ lies outside the tetrahedron. This test can be applied to points on the quadratic patch. From (5.1), the coordinates of points on the patch can be expressed as

$$\mathbf{x} = \sum_{j=1}^{6} \phi_j(\mathbf{X})\mathbf{x}_j = \sum_{j=1}^{6} \left(\mathbf{R}^T \mathbf{P}_j \mathbf{R}\right) \mathbf{x}_j,$$

where the matrix representation for $\phi_j(\mathbf{X})$ was introduced in Section 5.1.1. Substituting this expression into the inside/outside condition for each face, (5.2), results in

$$\mathbf{R}^T \left(\sum_{j=1}^{6} (\mathbf{x}_j \cdot \mathbf{n}_f)\mathbf{P}_j \right) \mathbf{R} - \mathbf{q}_f \cdot \mathbf{n}_f \quad \leq \quad 0.$$

Defining $\mathbf{E}_1 = [0,0,0;0,0,0;0,0,1]$ and using the fact that $\mathbf{R} = [X, Y, 1]^T$, the above inequality can be expressed as a quadratic form,

$$\begin{aligned} \mathbf{R}^T \mathbf{S}_f \mathbf{R} \quad &\leq \quad 0, \\ \mathbf{S}_f \quad &\equiv \quad \sum_{j=1}^{6} (\mathbf{x}_j \cdot \mathbf{n}_f)\mathbf{P}_j - (\mathbf{q}_f \cdot \mathbf{n}_f)\mathbf{E}_1. \end{aligned} \tag{5.3}$$

A quadratic form in this case is a polynomial of degree 2 in X and Y. (5.3) must be satisfied for all four faces of the tetrahedron in order for a patch reference point, entering the equation through $\mathbf{R} = [X, Y, 1]^T$, to map to the interior or to the surface of the tetrahedron.

In reference space, the set of points X, Y satisfying the quadratic form $\mathbf{R}^T \mathbf{S}_f \mathbf{R} = 0$, if not null, defines a conic section, or just "conic" for short. A more familiar expression for a conic is

$$\mathbf{R}^T \mathbf{S}_f \mathbf{R} = 0 \quad \Leftrightarrow \quad AX^2 + BXY + CY^2 + DX + EY + F = 0, \tag{5.4}$$

obtained by writing $\mathbf{S}_f$ as

$$\mathbf{S}_f = \begin{bmatrix} A & B/2 & D/2 \\ B/2 & C & E/2 \\ D/2 & E/2 & F \end{bmatrix}.$$

The set of points (X, Y) satisfying (5.4), if not null, defines one conic: an ellipse, a hyperbola, a parabola, a pair of lines, a single line, or a point. Figure 5-6 shows the set of conics associated with faces 1, 3, and 4 resulting from the tetrahedron-patch intersection in Figure 5-5. This figure is in the quadratic patch reference space, so that the unit right triangle denotes the extent of the patch. In this case, the conics associated with faces 1 and 3 are both hyperbolas, while the conic associated with face 4 is a pair of lines. The second line as well as the other halves of the hyperbolas are not shown as they do not intersect the reference triangle.

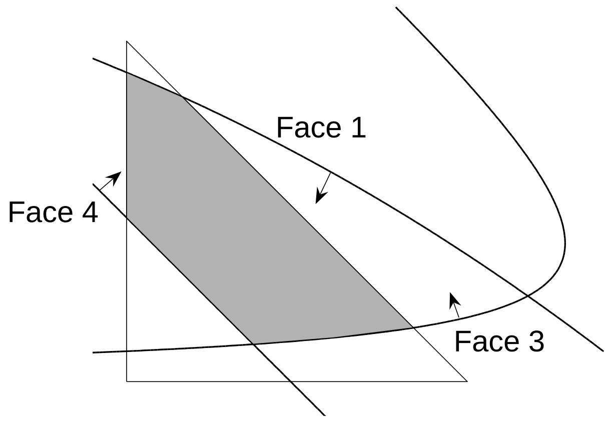


Figure 5-6: Intersection of the quadratic patch and tetrahedron from Figure 5-5, shown in the reference space of the patch. The plane of each tetrahedron face yields a conic in reference space. The arrows on the conics indicate the tetrahedron-interior direction on the patch. The shaded area is the patch region of validity and corresponds to the portion of the patch lying inside the tetrahedron.

As (5.3) is an inequality, points X, Y to one side of each conic will be valid, meaning that mapped into physical space, they will lie on the tetrahedron-interior side of the face corresponding to the conic. In Figure 5-6, the arrows indicate these directions of validity. The shaded area represents the boolean intersection of these regions inside the reference triangle. This area will be referred to as the region of validity of the patch with respect to the tetrahedron. Note, any point outside the reference triangle is no longer on the patch.

Construction of Cut Cells

The fact that the intersection between a tetrahedron face and a quadratic patch can be expressed as a conic in the patch reference space is an enabling feature that makes the cutting algorithm tractable. This section presents an overview of the cutting algorithm geared for these conic intersections. Details on intersection and parametrization methods are given in the next sections.

The cutting algorithm proceeds by looping over all tetrahedra in the background mesh. For each tetrahedron, bounding-box checks are performed to determine which patches, if any, can intersect the tetrahedron. A patch whose bounding box does not intersect the bounding box of the tetrahedron is not considered for intersection. The remaining patches are intersected with the tetrahedron according to the algorithm below.

Cutting algorithm for one tetrahedron

- Initialize I_{edge}, I_{face}, and F to null, where:
  I_{edge} = for each edge of the tetrahedron, the set of intersection points between the edge and the patches.
  I_{face} = for each face of the tetrahedron, the set of intersection points between the face and patch edges.
  F = set of 2D structures that will become the faces of the new cut cells.

- Loop over patches whose intersection with the tetrahedron is not precluded by the bounding-box test. For each patch:

  – Intersect patch with each of the four faces of the tetrahedron, yielding four conics, written in matrix form as $\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3$, and $\mathbf{S}_4$.

  – Determine 1D structures enclosing patch regions of validity:

    * Identify all (X, Y) intersections among the four conics and the three patch reference triangle edges. These include conic-conic intersections, conic-edge intersections, and edge-edge intersections (i.e. triangle vertices). For example, in Figure 5-6, eleven such intersections are shown.

    * Discard intersections lying outside the tetrahedron or not on the patch. An intersection, $R = [X, Y, 1]^T$, lies outside the tetrahedron if $\mathbf{R}^T\mathbf{S}_f\mathbf{R} > 0$ for one of the conics, $\mathbf{S}_f$, not part of the intersection. An intersection is not on the patch if (X, Y) is outside the unit reference triangle. For example, in Figure 5-6, five intersections are valid: one conic-conic intersection and four conic-line intersections. These intersections are circled in Figure 5-7. An example of an intersection that was discarded is the conic-conic intersection corresponding to faces 1 and 3, as it lies outside the reference triangle.

    * Check if any of the $\mathbf{S}_f$ are ellipses completely contained within the patch, as demonstrated in Figure 5-8. Such cases are identified by picking a point on each ellipse with zero intersections and checking if it lies inside the patch reference triangle. For data storage and integration purposes, the completely-

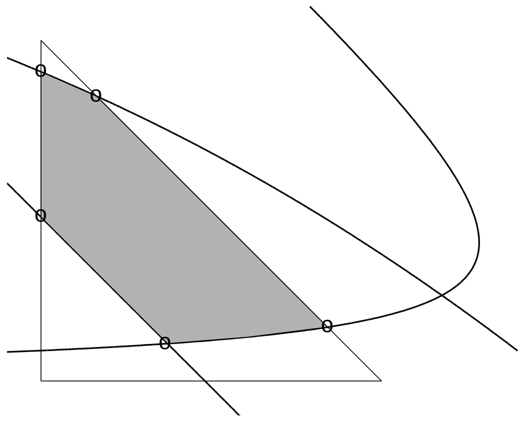


Figure 5-7: Valid intersections in patch reference space for the set of conics and patch edges shown in Figure 5-6. These five intersections (circled) consist of four conic-line intersections and one conic-conic intersection.

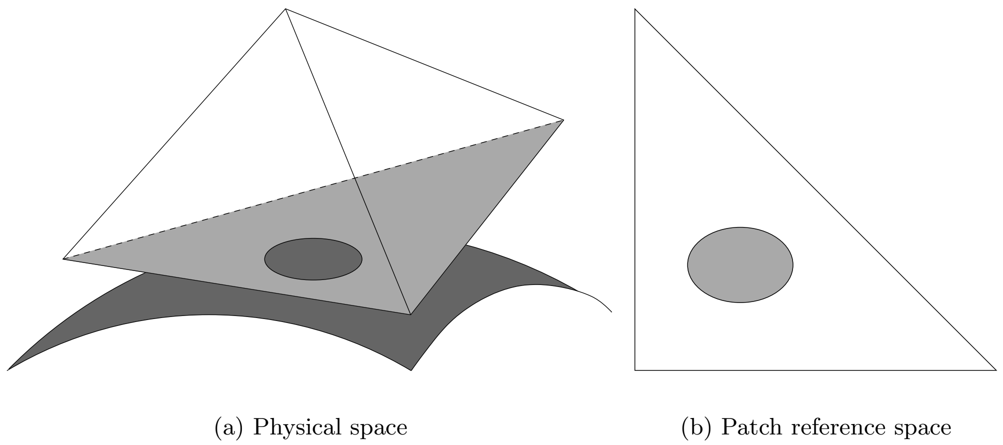


Figure 5-8: Intersection of a tetrahedron face with a curved patch resulting in an ellipse completely contained in the reference triangle. The validity region, shown shaded on the right, consists of the area inside the ellipse. Mapped into physical space, the associated embedded face is the portion of the patch inside the tetrahedron.

contained ellipses are divided in half by adding two artificial intersections on the major-axis extrema of the ellipse.

∗ If no valid intersections exist, the patch does not intersect the tetrahedron; continue to the next patch. Otherwise, store the valid intersections in I_{patch}.

∗ Parametrize each conic and assign a parameter value to every valid intersection on the conic. Similarly, parametrize each patch edge (e.g. 0 to 1) and assign a parameter to every intersection on the patch edge. For each conic and patch edge, arrange the intersections in order of increasing parameter.

        * Using conic normals and tangents, determine the direction of validity of each intersection with respect to the associated conic or edge. Store valid portions of each conic and edge in E_{patch}, which is the list of 1D structures specific to the patch.
        * Map conic-conic intersections from I_{patch} into physical space. These points lie on the patch and on both faces associated with the conics. Hence, they represent cuts on the tetrahedron's edges. Add each such intersection to the appropriate list, I_{edge}, and store the direction of validity and information on the adjacent patch 1D structures from E_{patch}.
        * Add intersections between conics and patch triangle edges (from I_{patch}) to the appropriate list, I_{face}. Mapped into physical space, these intersections lie on the faces of the tetrahedron and connect conics associated with different patches. Note, patch-to-patch connectivity is required to enable such connections. In practice, this connectivity is provided implicitly, by requiring neighboring patches to point to a unique set of linear node numbers when specifying the embedded geometry.
    - Tie together the patch 1D structures in E_{patch} into regions of validity using the algorithm presented in Section 4.1.2. The intersections in I_{patch} provide the connectivity information. Multiple disjoint regions bounded by distinct loops of 1D structures are possible and occur when a patch enters an element more than once. Each disjoint region of validity on a patch, mapped into physical space, is labeled as an *embedded face*.
    - Add the embedded faces to the list of 2D structures, F. For each enclosing 1D structure in E_{patch}, store the index of the adjacent embedded face, to be used for tying the 2D structures together.

- On each cut tetrahedron edge, order the intersections in I_{edge} according to distance along the edge. Note, these intersections were obtained from the mapped conic-conic intersections, as described above. Using validity directions at the intersections, identify portions of each edge lying inside the computational domain, and store these 1D structures in the list E_{edge}. In the process, flag the four vertices of the tetrahedron as lying inside or outside the computational domain.

- Construct regions of validity on each face of the tetrahedron. Begin by looping over the four faces:

    - Assemble E_{face} = list of relevant 1D structures associated with the face. This list includes members of E_{edge} for each edge adjacent to the face, as well as those conics in E_{patch} arising from patch-face intersections with this face. Note, 1D structures from contained ellipses are included in this list.

  - Tie together the 1D structures in E_{face} using the algorithm presented in Section 4.1.2. Tetrahedron vertices provide connectivity among 1D structures from E_{edge}. Intersections in I_{edge} provide connectivity between members of E_{edge} and E_{patch}. Finally, intersections in I_{face} provide connectivity among the conic 1D structures in E_{patch}. Note, each 1D structure stores several indices for its endpoints to keep track of all of this connectivity information. Each disjoint region of validity is labeled as a *cut face*.
  - Add the cut faces to the list of 2D structures, F. Store the index of each cut face with the adjacent 1D structures from E_{face} to be used for tying the 2D structures together.
  - If the face is an interior face of the background mesh, store information about the newly-created cut face(s) to provide connectivity between adjacent cut cells.

- Tie together the 2D structures in F, resulting in one or more sets of connected 2D structures. These sets constitute bounded volumes that are the new cut cells. The algorithm for performing this tying proceeds similarly to the one in Section 4.1.2, except that 1D structures in E_{patch} and E_{edge}, as opposed to intersection points, now provide the connectivity information.

An example demonstrating the key points of the above algorithm is illustrated in Figure 5-9 for the cut cell introduced in Figure 5-4. An intersection between each patch and the tetrahedron yields patch edge 1D structures and conic 1D structures enclosing embedded faces. Note, at least one embedded face is created for each intersecting patch. In this example, there are six intersecting patches and six embedded faces. Three of the patches contain valid conic-conic intersections that map to cuts on the edges of the tetrahedron. These cuts are used to construct the three tetrahedron-edge 1D structures. Three of the four faces of the tetrahedron are cut, yielding, in this case, three cut faces. Altogether, the six embedded faces and the three cut faces enclose one cut cell.

Another cutting example is illustrated in Figure 5-10, where a tetrahedron is cut by two quadratic patches. While the tetrahedron in the previous example produced one cut cell, two cut cells are created in this example. In the figure, portions of the tetrahedron inside the computational domain are the two right-hand-side corners, each cut by a single patch. The resulting twelve 1D structures are shown in Figure 5-10b: six come from mapped

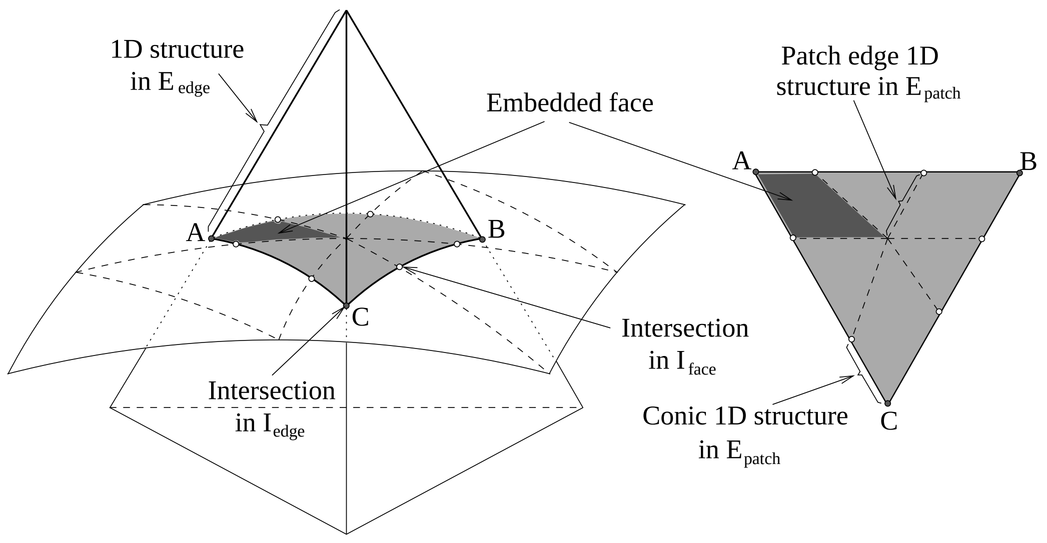
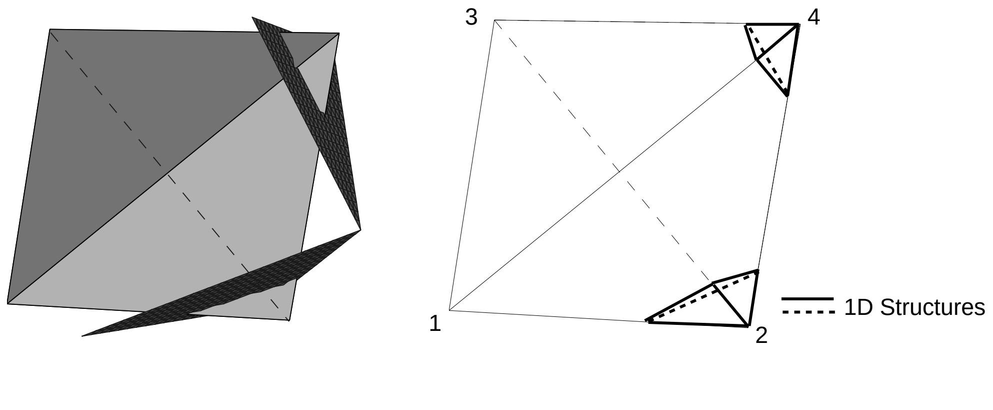

1D structure in E_{edge}
Embedded face
Patch edge 1D structure in E_{patch}
A
B
C
Intersection in I_{face}
Intersection in I_{edge}
Conic 1D structure in E_{patch}

Figure 5-9: Detailed intersections for the cut cell introduced in Figure 5-4. On the right is a top-view of the portion of the quadratic-patch surface contained within the tetrahedron. Various intersection points and 1D structures are indicated. One out of the six embedded faces is highlighted.

conics, and the other six come from cut portions of the tetrahedron's edges. These 1D structures bound eight 2D structures: two embedded faces and six cut faces. Tying these 2D structures results in two disjoint sets, bounding the resulting two cut cells. As in two dimensions, multiple cut cells arising in this fashion are treated separately, each one with its own solution approximation.



(a) Patch intersections

(b) Resulting 1D structures

Figure 5-10: Two cut cells arising from the cutting of one tetrahedron. The two corners adjacent to nodes 2 and 4 of the original tetrahedron lie inside the computational domain. In addition, two of the original tetrahedron's faces are cut into multiple (two) disjoint regions.

Background mesh tetrahedra completely contained in the embedded geometry are identified and removed from the computational mesh structure. The identification process is similar to that used in two dimensions. During the cutting, nodes lying outside the computational domain are identified when ordering the edge cuts. For example, nodes 1 and 3 in Figure 5-10 lie outside the computational domain. This information is propagated to other nodes by traversing uncut tetrahedra in the background mesh and setting the inside/outside flag to the same value for all the nodes of each uncut tetrahedron. Uncut tetrahedra with adjacent nodes outside the computational domain must lie outside the computational domain, and hence are removed. Similarly, uncut background-mesh faces with adjacent nodes outside the computational domain are also removed.

Intersections Involving Conics

Crucial to the cutting algorithm is the ability to robustly intersect conics with the reference triangle edges and with each other. As discussed in the presentation of the cutting algorithm, these intersections are required for constructing the 1D structures enclosing the patch regions of validity in reference space. First, intersecting a conic with a line segment is straightforward. The line segment can be written in parametric form, $X(t) = X_1 + t(X_2 - X_1), Y(t) = Y_1 + t(Y_2 - Y_1)$, where (X_1, Y_1) and (X_2, Y_2) are the line endpoints, and $t \in [0, 1]$ is the parameter. Substituting these expressions for X and Y into the quadratic form $\mathbf{R}^T\mathbf{S}_f\mathbf{R} = 0$ yields a quadratic equation for t. Solving this equation gives at most two possible values for t, each of which corresponds to an ordered pair (X, Y). Solutions $t < 0$ and $t > 1$ are discarded as they do not lie on the segment. The quadratic equation obtained in this fashion is suitable for numerical computation in that double roots for t always correspond to tangency conic-line intersections. Such numerical consideration becomes increasingly important for conic-conic intersections.

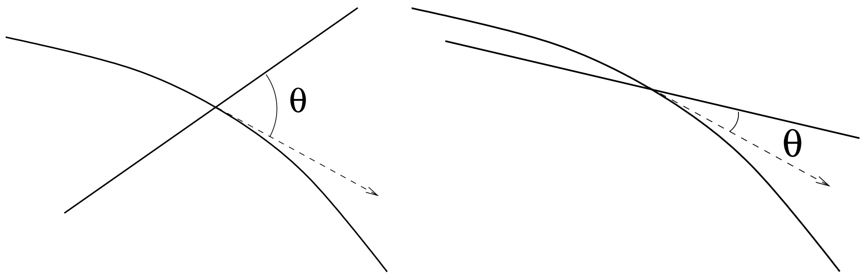
Compared to a conic-line intersection, robustly intersecting two conics is more involved. Consider two conics arising from $\mathbf{R}^T\mathbf{S}_1\mathbf{R} = 0$ and $\mathbf{R}^T\mathbf{S}_2\mathbf{R} = 0$. The task is to find all points (X, Y) that lie on both of the conics. One possible analytical solution strategy, not used in this work, begins by determining the linear combination $\mathbf{S}_3 = a\mathbf{S}_1 + b\mathbf{S}_2$ that eliminates (for example) the X^2 term. The equation $\mathbf{R}^T\mathbf{S}_3\mathbf{R} = 0$ is then solved to obtain X as an at-most second-degree polynomial in Y, and this equation for X is substituted back into either of the original quadratic forms, yielding a quartic equation in Y. While this approach is mathematically correct, it is not robust in finite precision arithmetic. For example, for multiple non-tangency intersections sharing the same Y coordinate, the quartic equation exhibits a double-root tangency, which could easily be missed due to machine-precision errors in calculating the quartic coefficients.

An alternate conic-conic intersection strategy, used in this work, is based on the idea of finding one or more degenerate conic sections that are linear combinations of $\mathbf{S}_1$ and $\mathbf{S}_2$ [37]. The method proceeds by finding t values that result in the matrix $\mathbf{S}_3 = t\mathbf{S}_1 + (1-t)\mathbf{S}_2$ having zero determinant. Writing out the determinant of the 3×3 matrix $\mathbf{S}_3$ results in a cubic equation for t, guaranteeing at least one real root. Up to a multiplicative constant, the only linear combination of $\mathbf{S}_1$ and $\mathbf{S}_2$ not represented by the above form for finite t is $\mathbf{S}_3 = \mathbf{S}_1 - \mathbf{S}_2$. Hence, this matrix is also checked for zero determinant. Each resulting zero-determinant $\mathbf{S}_3$ represents a degenerate conic (i.e. one or more lines). This can be verified analytically by rotating the coordinates to a frame where the coefficient of XY in $\mathbf{S}_3$ is zero, as will be demonstrated in (5.6), writing out the determinant of $\mathbf{S}_3$, and considering each possible degeneracy in turn: a pair of horizontal lines, a pair of vertical lines, or a pair of crossing lines. Each possible zero-determinant case can be paired with one of the degenerate conics. Intersecting the lines from the degenerate conic with either of the conics $\mathbf{S}_1$ or $\mathbf{S}_2$ yields exactly the required intersections. This is because if, for a given (X, Y), both $\mathbf{R}^T\mathbf{S}_1\mathbf{R} = 0$ and $\mathbf{R}^T\mathbf{S}_2\mathbf{R} = 0$, then necessarily $\mathbf{R}^T\mathbf{S}_3\mathbf{R} = 0$, as $\mathbf{S}_3$ is a linear combination of $\mathbf{S}_1$ and $\mathbf{S}_2$. Thus, intersections between $\mathbf{S}_1$ and $\mathbf{S}_2$ also lie on $\mathbf{S}_3$. Similarly if $\mathbf{R}^T\mathbf{S}_3\mathbf{R} = 0$ and $\mathbf{R}^T\mathbf{S}_1\mathbf{R} = 0$, then $\mathbf{R}^T\mathbf{S}_2\mathbf{R} = 0$, and vice-versa. Hence, no extraneous intersections are possible. In practice, the lines from $\mathbf{S}_3$ are intersected with both $\mathbf{S}_1$ and $\mathbf{S}_2$, and the numerically best-conditioned set of intersections is chosen. Numerical conditioning of a set of intersections is determined by the smallest intersection angle in the set, where the intersection angle is the acute angle between the tangent vectors of two intersecting curves. A large intersection angle is better-conditioned compared to a small intersection angle, as illustrated in Figure 5-11. The conic-conic intersection strategy based on line degeneracies has been found to be much more robust than the quartic solving technique discussed above. To improve accuracy of the obtained intersection points, a Newton-Raphson method is applied to the equations $\mathbf{R}^T\mathbf{S}_1\mathbf{R} = 0$ and $\mathbf{R}^T\mathbf{S}_2\mathbf{R} = 0$.

Conic Parametrization

The cutting algorithm requires ordering intersections on a conic in terms of a single parameter for identifying 1D structures in patch reference space. This ordering requires the parametrization of each conic. Specifically, of interest is the parametrization of an ellipse, a parabola, and a hyperbola. As shown in (5.4), the equations for these conics in patch reference space can be written as

$$AX^2 + BXY + CY^2 + DX + EY + F = 0. \tag{5.5}$$



(a) Large intersection angle (b) Small intersection angle

Figure 5-11: Numerical calculation of the intersection between two curves is better conditioned for a large intersection angle, θ, (a) compared to a small intersection angle (b). In (b), the set of points a distance ϵ apart between the two curves has diameter $>> \epsilon$, and hence the precise location of the intersection is more difficult to identify.

Each conic is classified as either a parabola, hyperbola, ellipse, or a degenerate form based on the value of the discriminant, $\delta = B^2 - 4AC$. The conic is a hyperbola if $\delta > 0$, a parabola if $\delta = 0$, and an ellipse (or degenerate) if $\delta < 0$. As degenerate forms are lines, their parametrization is straightforward. For parametrization of the non-degenerate conics, the coordinates are rotated to eliminate the cross term, XY. In particular, setting

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} X' \\ Y' \end{bmatrix}, \qquad \alpha = \tan^{-1}\left(\frac{B}{C-A}\right), \tag{5.6}$$

the conic equation (5.4) becomes

$$A'X'^2 + C'Y'^2 + D'X' + E'Y' + F' = 0, \tag{5.7}$$

where

$$\begin{aligned} A' &= A\cos^2(\alpha) - B\sin(\alpha)\cos(\alpha) + C\sin^2(\alpha), \\ C' &= A\sin^2(\alpha) + B\sin(\alpha)\cos(\alpha) + C\cos^2(\alpha), \\ D' &= D\cos(\alpha) - E\sin(\alpha), \\ E' &= D\sin(\alpha) + E\cos(\alpha), \\ F' &= F. \end{aligned}$$

In this rotated coordinate system, all of the non-degenerate conics can be parametrized by angle, θ, from a single point, (X'_f, Y'_f), according to

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} X'_f + r(\theta)\cos(\theta) \\ Y'_f + r(\theta)\sin(\theta) \end{bmatrix}, \tag{5.8}$$

where $r(\theta)$ is the distance to the conic from (X'_f, Y'_f), with the angle measured in standard fashion counterclockwise from the horizontal. The parametrization of each of the conics is presented in Appendix E. In general, a conic focus is used for (X'_f, Y'_f), except for custom cases in which the focus is too far from the region of interest to allow for a well-conditioned parametrization. This custom parametrization is also outlined in the Appendix. Parametrization of periodic conics (i.e. contained ellipses) does not pose a problem because these conics are split and each half is parametrized separately.

5.1.3 Integration

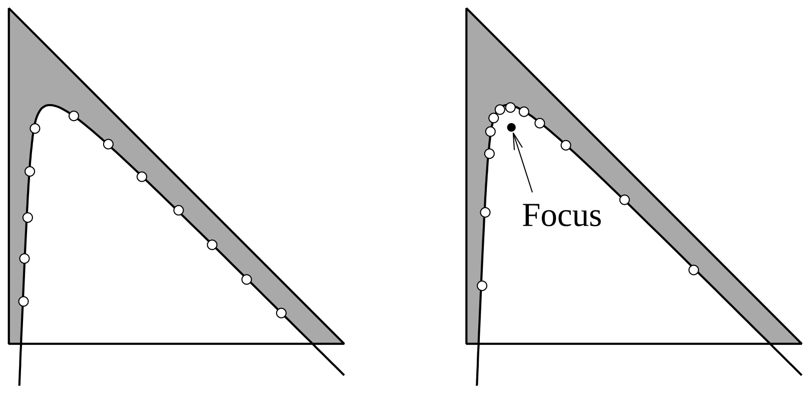
As in two dimensions, irregularly-shaped cut cells require a modified integration technique since tetrahedral-based quadrature rules are no longer valid. Fortunately, the basic idea developed for two-dimensional cut cells, involving sampling point "speckling" and the divergence theorem, extends naturally to volume integration in three dimensions. The two-dimensional integration technique is in fact also used directly in three dimensions for deriving surface integration rules on embedded faces and cut faces. The following sections describe the derivation of the volume and surface integration rules, paying careful attention to numerical conditioning, which becomes particularly important in three dimensions.

Conics

Deriving volume integration rules on the cut cells requires the ability to integrate on the bounding 2D structures. As the 2D structures are themselves irregular shapes, integration rules must also be derived on them, thus requiring the ability to integrate on the bounding 1D structures. As discussed in Section 5.1.2, 1D structures arise either from segments of straight lines or from segments of possibly-curved conics. High-order accurate integration on these structures is achieved by mapping each one to a reference interval and using Gaussian quadrature on the interval. While this mapping is trivial for straight segments, mapping a curved conic segment requires a well-conditioned parametrization.

Parametrization of a conic for integration need not be the same as the one presented in the previous section for use in ordering intersections in the cutting algorithm. For an ideal integration parametrization, the integrand should vary smoothly with the parameter, so that numerical integration with a fixed number of Gauss points is accurate. For example, parametrization by arc length along the conic is not suitable, as insufficient points may be placed in areas of high curvature – note, conic integrations will involve the conic normal, which varies greatly in these areas.

From experience, the conic parametrization used in the cutting algorithm is reasonably well-suited for integration. That is, in general, the conic normal vector varies relatively smoothly with the angle from a focus (or from a custom parametrization point). Therefore, this parametrization is used for integration in this work. An example of a situation when this parametrization may begin to lose accuracy is illustrated in Figure 5-12. The highlighted patch validity region is bounded by (portions of) the edges of the reference triangle and a conic which is one half of a hyperbola. The localized region of high curvature on the conic makes parametrization by arc length, shown in Figure 5-12a, highly inaccurate. Parametrization by angle from a focus, shown in Figure 5-12b, captures the region of high curvature, but places few points on the outer portions of the hyperbola legs. As the integrand in this area is sampled with fewer points, degradation in integration accuracy is possible. However, for the orders tested, and with a large number of 1D quadrature points (the default is 20), no associated accuracy problems have been observed thus far. More accurate integration parametrizations are certainly possible and could be the subject of future work.



(a) By arc length

(b) By angle from focus

Figure 5-12: Parametrization of a conic with a localized region of high curvature. Possible integration points are shown for two cases. Points obtained from a parametrization by arc length (a) fail to adequately capture the area of high curvature. Points from a parametrization by angle from a focus (b) capture the curvature; however, fewer sampling points away from the region of high curvature could lead to loss of accuracy for high-order integrands.

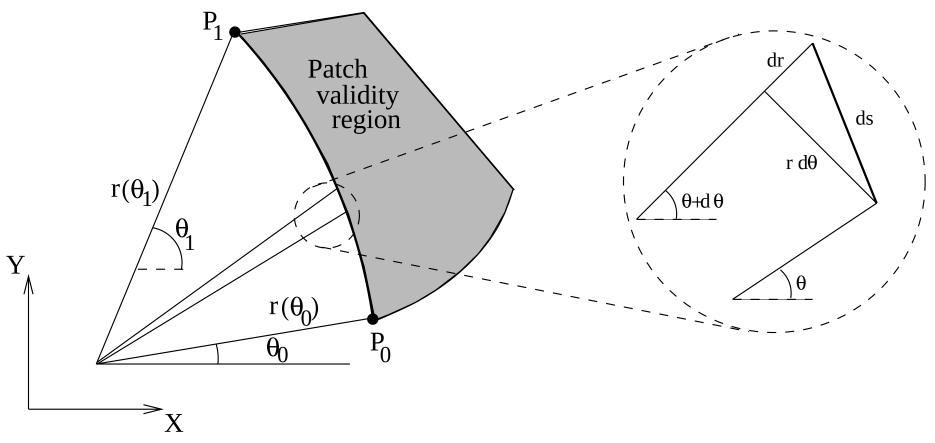


Figure 5-13: Integration along a curved conic segment adjacent to a patch validity region is performed by parametrizing the conic and using Gaussian quadrature on the parameter interval. Shown in this figure is a parametrization by angle from a point (e.g. a focus of the conic).

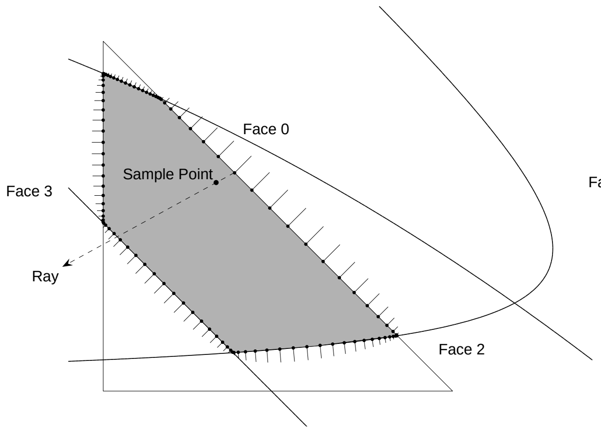
Integration using the angle parametrization proceeds as follows. For each curved conic segment adjacent to a patch validity region, the two intersection points bounding the conic segment are labeled P_0 and P_1, as shown in Figure 5-13. The X, Y coordinates of P_0 and P_1 are converted to polar form according to the parametrization, yielding a mapping from a θ interval, $[\theta_0, \theta_1]$ to the curved conic. An integral from P_0 to P_1 along the conic segment of an arbitrary function, $f(\mathbf{X}, \mathbf{n})$, can therefore be written as

$$\int_{P_0}^{P_1} f(\mathbf{X}, \mathbf{n}) ds = \int_{\theta_0}^{\theta_1} f\left(\mathbf{X}(\theta), \mathbf{n}(\theta)\right) \left|\frac{ds}{d\theta}\right| d\theta, \qquad \left|\frac{ds}{d\theta}\right| = \sqrt{\left(\frac{dr}{d\theta}\right)^2 + r^2},$$

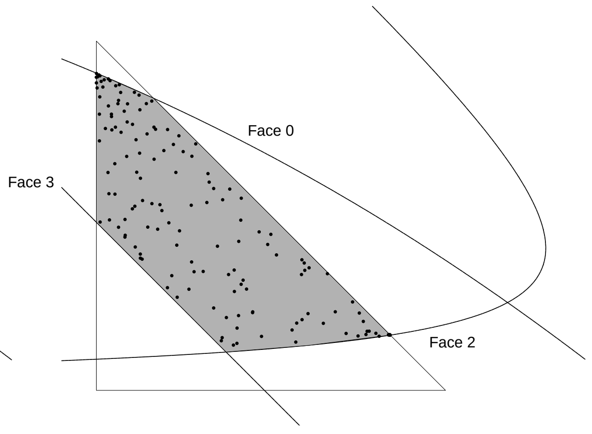
where the differential arc length is illustrated in the inset in Figure 5-13. Numerical integration is achieved in standard fashion via Gauss quadrature on the $[\theta_0, \theta_1]$ interval. $dr/d\theta$ is computed by differentiating the parametrization, $r(\theta)$. The outward-pointing normal vector on the conic, $\mathbf{n}$, is obtained directly by taking the gradient of the Cartesian representation in (5.4). This gradient is outward-pointing as the set of points for which $\mathbf{R}^T\mathbf{S}\mathbf{R}$ is positive lies outside the region of validity. An example of the quadrature points and normals on the 1D structures adjacent to the validity region in Figure 5-6 is shown in Figure 5-14a.

Embedded Faces

The integration rules for 1D structures derived in the previous section allow for the application of the two-dimensional integration technique (Section 4.1.3) to the validity region in reference space. Recalling that the validity region represents the portion of a patch within



(a) 1D integration points and normals



(b) 2D integration points

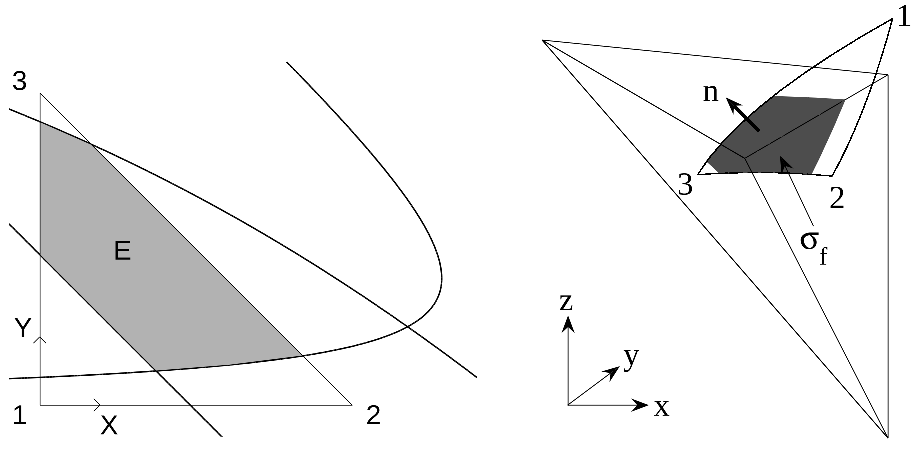
Figure 5-14: Patch reference space: quadrature points and normals on 1D structures adjacent to validity region (a) and embedded face quadrature points obtained by ray-casting (b). The outward-pointing normals in (a) are illustrated by line segments whose length is weighted by the Gauss quadrature weights and by the mapping Jacobian $|ds/d\theta|$.

a given tetrahedron, these rules will allow for integration on the surface of the embedded geometry inside the computational domain. As mentioned in the previous section, each validity region mapped to physical space becomes an individual embedded face. A typical integral on an embedded face, σ_f, transforms to an integral on the reference validity region, E, as follows:

$$\int_{\sigma_f} f(\mathbf{x},\mathbf{n})dA = \int_E f(\mathbf{x}(\mathbf{X}),\mathbf{n}(\mathbf{X}))\,|\mathbf{J}(\mathbf{X})|\,dE, \tag{5.9}$$
$$\mathbf{J}(\mathbf{X}) = \frac{d\mathbf{x}}{dX}\times\frac{d\mathbf{x}}{dY}, \quad \mathbf{n}(\mathbf{X}) = -\frac{\mathbf{J}(\mathbf{X})}{|\mathbf{J}(\mathbf{X})|}.$$

The derivatives $d\mathbf{x}/dX$ and $d\mathbf{x}/dY$ used in the mapping Jacobian, $J(\mathbf{X})$, are obtained readily from the patch surface description, (5.1). By the patch node ordering convention introduced in Section 5.1.1, the vector $\mathbf{J}(\mathbf{X})$ points into the computational domain. Hence, the normal vector pointing out of the computational domain, $\mathbf{n}(\mathbf{X})$, is defined in the direction of $-\mathbf{J}(\mathbf{X})$. The mapping of E to σ_f is illustrated in Figure 5-15, where the three patch vertices are labeled in both the reference space and the physical space. The normal vector, $\mathbf{n}$, points out of the computational domain.

By the transformation in (5.9), integration on curved embedded faces reduces to integration on the planar validity regions in patch reference space. The integration technique



(a) Patch validity region (b) Embedded face

Figure 5-15: Relation between patch validity region, E, in reference space (X, Y), and the mapped embedded face, σ_f, in physical space (x, y, z).

developed for two-dimensional cut cells applies directly in this case. Specifically, an integration basis is defined on a fitted bounding box of the reference validity region, and the divergence theorem is used to convert interior area integrals into boundary integrals on the 1D structures. Random sampling points in the interior of each validity region are obtained by ray casting from randomly-chosen 1D quadrature points, as illustrated in Figure 5-14a. In order to avoid clustering in areas near short 1D segments, the probability of choosing a starting quadrature point is weighted by the length of its 1D segment. Determining the exit point of each ray out of the validity region requires solving conic-line intersection problems, as described in Section 5.1.1. A typical set of resulting sampling points inside the validity region is shown in Figure 5-14b.

Cut Faces

The two-dimensional integration technique of Section 4.1.3 is also used to derive integration rules on faces cut by the patch geometry. As these cut faces are already planar, no reference-space transformation, such as that used for embedded faces, is necessary. An integration basis is defined on a fitted bounding box of each cut face. Integration on the 1D structures enclosing the cut faces requires care for curved conic segments. Since the face validity region lies in physical space, the conic segment quadrature weights are modified to include the mapping from reference space to physical space. Outward-pointing normal vectors along the curved conic segments are obtained by projecting the physical patch normals onto the plane of the face. By virtue of the existence of the intersection, the physical patch normal must have a component in the plane of the face pointing into the computational domain. The outward-pointing normal is then taken in the negative direction of this projected component. Sampling point selection is performed in the coordinates of the plane containing the face, using ray-casting with length-weighted selection of the starting 1D quadrature points. Since an analytical representation of the conic 1D structures in physical space is not readily available, testing for the point of exit of each ray requires performing intersections in physical space between the ray and the patches intersecting the element. Intersecting a line with a patch in three-dimensional space amounts to solving a conic-conic intersection problem in the patch reference space. Specifically, expressing the line as $\mathbf{x} = \mathbf{x}_0 + (\Delta\mathbf{x})t$ and substituting into (5.1) yields three equations (one for each coordinate) in three unknowns: t, X, Y. As t appears linearly, it can be eliminated, leaving two quadratic forms in X and Y. The solution is then found using the conic-conic intersection method described in Section 5.1.2.

Figure 5-16 illustrates a cut face resulting from cutting the tetrahedron in Figure 5-4. This cut face is bounded by five 1D structures: two from E_{edge} (CD and BD), and three from E_{patch} (between B and C). Sampling points for this cut face, generated by ray-casting, are also shown in the figure. During ray-casting, the point of exit is determined by performing patch-line intersections with each of the three intersecting patches in addition to line-line intersections with the original face edges.

Cut Cells

Integration on three-dimensional cut-cell interiors is performed using an extension of the two-dimensional method. That is, for each element, κ, sampling points, x_q, and weights, w_q are found for integrating arbitrary $f(\mathbf{x})$ up to a specified order of accuracy, according to

$$\int_\kappa f(\mathbf{x})d\mathbf{x} \approx \sum_q w_q f(\mathbf{x}_q).$$

The derivation of quadrature weights presented in Section 4.1.3 for two dimensions ($d = 2$) is also valid for three dimensions ($d = 3$). In particular, for a given set of sampling points x_q, the w_q are given by (4.4), where $\zeta_{\mathbf{i}}(\mathbf{x}) = \partial_k G_{\mathbf{i}k}$, and the $G_{\mathbf{i}k}(\mathbf{x})$ are defined as in (4.3). The bounding box used in defining the $G_{\mathbf{i}k}(\mathbf{x})$ is now three-dimensional, and the $\Phi_{\mathbf{i}}(\mathbf{x})$ are tensor products of three one-dimensional Lagrange basis functions. The order of these functions determines the accuracy of the integration. For the 3D compressible Navier-Stokes equations, in which the integrands are generally not polynomial, an integration

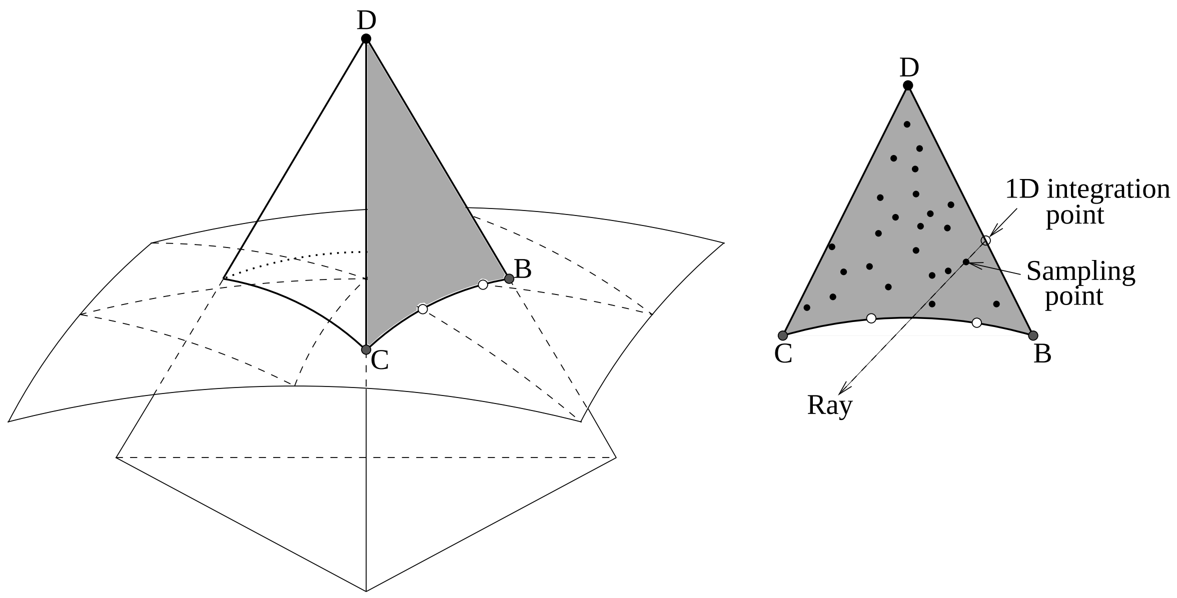


Figure 5-16: Three cut faces result from the intersection depicted in Figure 5-4. This figure highlights one of these cut faces. Sampling points generated by ray-casting in the plane of the cut face are also shown.

order of $2p + 1$, where p is the solution interpolation order, has been found to be sufficient. Integrands of the $G_{\mathbf{i}k}(\mathbf{x})$ on the element boundary, required for the computation of the weights, are performed using the 2D integration rules derived in the previous sections for cut faces and embedded faces.

Sampling points, x_q, inside κ are found using ray-casting from the element boundary surface integration points. The surface points for the ray origins are chosen at random, using an area-weighted selection method giving preference to larger areas. As in two dimensions, this area-weighting prevents clustering of points cast from quadrature points on very small cut or embedded faces. The rays are cast into the element along inward-pointing normal directions, perturbed randomly by up to 15^o. The point of first exit of these rays is determined by intersecting them with the faces of the original, background tetrahedron and with the intersecting patches. The line-patch intersection problem is solved as described in the previous section on cut-face integration. For each ray, a random sampling point is chosen between the ray origin and the point of first exit out of the element. Figure 5-17 shows an example of a set of interior sampling points generated by this process.

Numerical conditioning plays an important role in the creation of the 3D element-interior integration rules. Specifically, the weight calculation can become ill-conditioned when the cut-element bounding box used for defining the $G_{\mathbf{i}k}(\mathbf{x})$ functions does not tightly fit the element. An example of such a situation is shown in Figure 5-18a. The ill-conditioning

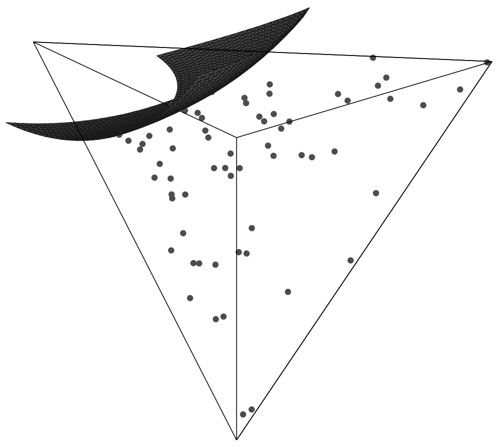
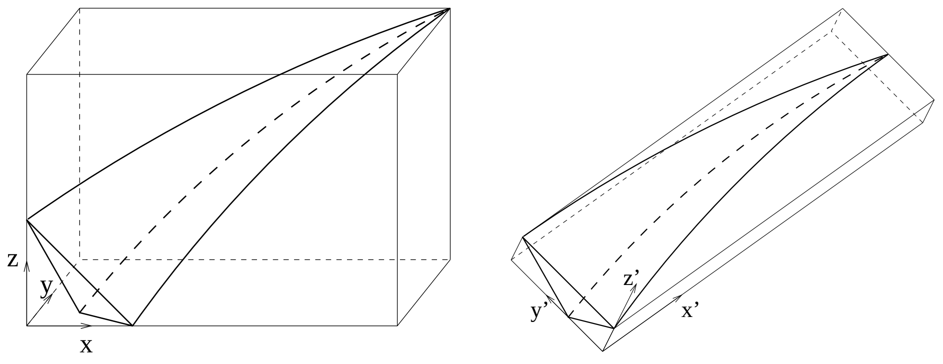


Figure 5-17: Interior, volume-integration sampling points for the cut tetrahedron depicted in Figure 5-5. These points were generated by ray casting from embedded-face and cut-face integration points.



(a) Axis-aligned bounding box.

(b) Oriented bounding box.

Figure 5-18: Numerical conditioning improvement of element-interior integration via bounding-box rotation for the case of a sliver element.

in such a case arises from the fact that many of the tensor-product Lagrange functions, $\Phi_{\mathbf{i}}(\mathbf{x})$, are associated with nodes far away from the element and hence appear very similar (near zero) over the element. This effect was observed in 2D, although in 3D it is more pronounced, appearing at lower interpolation orders and on cut cells that are not severely anisotropic. As in 2D, the solution to this conditioning problem consists of using a fitted, oriented, bounding box, such as the one shown in Figure 5-18b.

An oriented bounding box for a 3D cut cell is created based on the set of surface integration points, according to the following procedure. First, two points of maximum separation distance are chosen from the surface integration points, which include the points on the cut faces and on the embedded boundaries. The first direction of the oriented bounding box, $\hat{\mathbf{x}}'$, is chosen along the line connecting these two points. Next, all the points are projected to the plane orthogonal to $\hat{\mathbf{x}}'$. From the set of projected points, two of maximum separation distance are chosen, defining the second direction of the oriented bounding box, $\hat{\mathbf{y}}'$. Finally, the third direction, $\hat{\mathbf{z}}'$, is orthogonal to both $\hat{\mathbf{x}}'$ and $\hat{\mathbf{y}}'$. Applying this method to the sliver element of Figure 5-18a yields the oriented bounding box shown in Figure 5-18b.

The oriented bounding-box is used in the integration-rule derivation for all cut cells. For thin sliver elements, it greatly improves the numerical conditioning of the integration weight calculation, yielding much more accurate integration rules. However, for some elements, even with an oriented bounding box, the weight calculation may become ill-conditioned as the desired integration order increases. This problem is not unexpected, since even an oriented bounding box may not provide a tight fit to highly curved and/or anisotropic cut cells. In practice, the ill-conditioning has been observed for some cut-cell meshes when the integration accuracy exceeds fifth order, which corresponds to a compressible Navier-Stokes calculation using interpolation order $p \geq 3$. Improving the integration conditioning for higher orders is a possible area of future research.

5.1.4 Implementation

The three-dimensional cut-cell method was implemented in the same discontinuous Galerkin finite element code used for the two-dimensional cut-cell method. While the cutting algorithm is fundamentally different and more complex in three dimensions, this complexity is isolated to a pre-processing step and effectively hidden from other parts of the code. In particular, the output of the 3D cutting algorithm consists of quadrature points for cut volume and area integrations, and connectivity between embedded faces and adjacent cut cells. This information is used implicitly by higher-level functions in place of the standard reference quadrature rules and mesh connectivities. The data structures for storing the 3D cut-cell information are the same as the ones in the 2D implementation. Solution interpolation functions on cut cells are defined on shadow tetrahedra, taken to be the right tetrahedra associated with the oriented cut-cell bounding boxes.

In a 3D cut-cell adaptive run, the user specifies the initial background volume mesh and a quadratic-patch representation of the embedded geometry. The quadratic-patch representation consists of the patch node coordinates and a list consisting of six node indices per patch. Patch connectivity is determined by the linear nodes, which are the first three nodes in Figure 5-1. Thus, these nodes should not be repeated in the node list. The background volume mesh consists of linear tetrahedra enclosing the computational domain. Note, as mentioned in Section 5.1.2, the embedded geometry is not required to lie completely inside the background domain (e.g. for symmetry boundary conditions).

Output error estimation remains fundamentally unchanged from the 2D implementation. Minor modifications to the patch reconstruction algorithm were made to generalize it to three dimensions. Re-meshing of the computational domain occurs at every adaptation iteration, and, as in 2D, the metric for adaptation is specified on the background mesh. A grid-implied metric (Section 3.2.2) is used on background mesh elements completely contained within the embedded geometry. Adaptive meshing of the background mesh is performed using the TetGen mesh-generation package discussed in Section 3.2.3. Currently, this package supports isotropic refinement, in which metric definition amounts to specifying element volumes. L_2 projection is used to transfer the solution to the adapted mesh, with $p = 0$ restriction on cut cells.

5.2 Results

The 3D cut-cell method is applied to several representative aerodynamic cases. The goals of this section are to demonstrate the accuracy of the 3D cut-cell method and to verify convergence of the adaptive method. Boundary-conforming adaptive results are not shown due to the unavailability of a robust and automated three-dimensional mesher for curved geometries. In particular, the approach of post-processing linear boundary-conforming meshes to introduce surface curvature is prone to failure, even for isotropic meshes.

Comparisons of the adapted meshes and the error convergence histories are given in terms of degrees of freedom for the cut-cell method using interpolation orders $p = 0$ to $p = 2$. As in the 2D results, the DOF count does not include the equation-specific multiplier, such as 5 for the 3D Euler equations. Computational work estimates are also given with the DOF results. As described in Section 4.2, the work estimate is of the form $W \sim N_e(n(p))^a =$

DOF$(n(p))^{a-1}$, where N_e is the number of elements, $n(p)$ is the degrees of freedom per element, and a is a measure of the computational complexity. In three dimensions, $n(0) = 1$, $n(1) = 4$, and $n(2) = 10$. Thus, assuming again that $a \sim 2$, $p = 1$ is expected to be four times more expensive than $p = 0$ and $p = 2$ is expected to be two and a half times more expensive than $p = 1$.

5.2.1 Channel Flow Over a Gaussian Perturbation, $M = 0.3$

This case considers inviscid flow through a channel, the floor of which has a Gaussian perturbation in the streamwise (x) direction. The output of interest for the adaptive runs is the drag, which is the x-component of the force on the channel floor. Refinement is expected primarily in the vicinity of the bump, as the only contribution to the drag is pressure exerted on portions of the geometry that have a non-zero x-component to their normal. The true value of the drag for this flow is not exactly zero due to the proximity of the inflow and outflow boundaries. Thus, the true value is computed using $p = 3$ interpolation on a structured, 18432-element, boundary-conforming mesh, shown in Figure 5-19. This mesh was generated manually, and elements with faces or edges on the bottom surface were curved to cubic order, $q = 3$ [24]. Comparing the adapted cut-cell results to a boundary-conforming true value tests the accuracy of the cut-cell method.

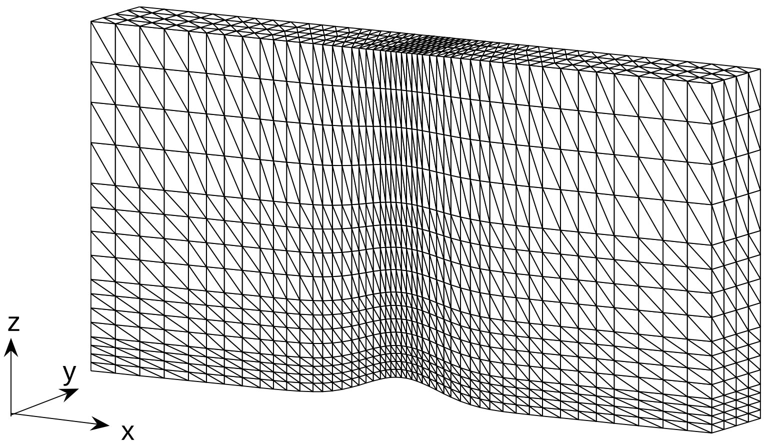


Figure 5-19: Gaussian bump channel: $M = 0.3$. Manually-generated, 18432-element, boundary-conforming mesh, with $q = 3$ curved elements on the bottom wall. The drag from a $p = 3$ solution on this mesh serves as the true value for the cut-cell adaptive runs.

The floor of the channel, which is the embedded surface in this case, is represented with 144 quadratic patches. 24 intervals are used in the x direction and 3 intervals are used in the y direction. The distribution is logarithmic in the x direction, with approximately 12 patches lengthwise across the bump. An initial background mesh for the adaptation was generated manually by subdividing rectangular parallelepipeds of a structured mesh. Figure 5-20 shows the initial 576-element background mesh as well as the curved embedded surface. The embedded surface is made slightly larger than necessary so that it protrudes out of the background domain. As discussed in Section 5.1.2, while the geometry outside the background domain is discarded during the cutting algorithm, extending the geometry in this manner prevents degeneracies when locating intersections.

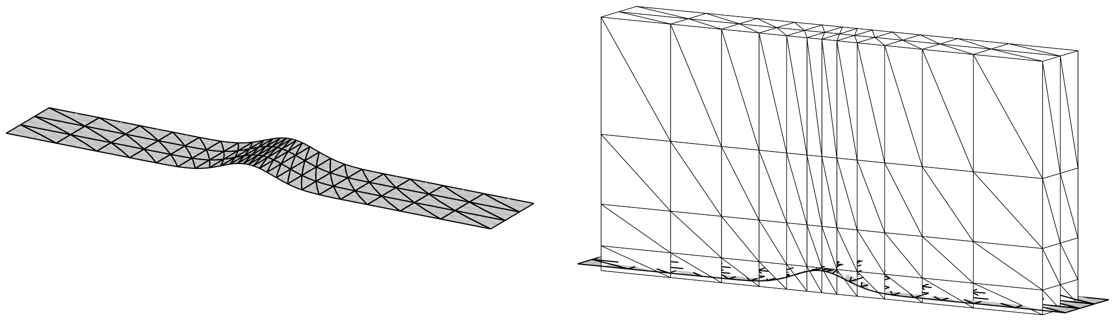

(a) 144 surface patches (b) 576-element initial mesh

Figure 5-20: Gaussian bump channel: $M = 0.3$. Quadratic patch representation of the bump surface (a) and the initial background mesh (b).

Adaptive runs were performed for $p = 0, 1, 2$, using 0.15 counts for the drag tolerance. The drag coefficient was computed using the floor planform as the reference area. Figure 5-21 shows the results of the adaptation runs. Adaptation for $p = 0$ was not continued down to the drag tolerance as computational costs became prohibitive. However, assuming that the $p = 0$ convergence rate continues, over 10^{10} DOF would be necessary to reach the error tolerance. On the other hand, both $p = 1$ and $p = 2$ converge to the desired error tolerance, at which point $p = 2$ requires an order of magnitude fewer degrees of freedom than $p = 1$. In terms of estimated work at the error tolerance, $p = 2$ is cheapest by a factor of four compared to $p = 1$ and by over four orders of magnitude compared to $p = 0$.

The final adapted meshes for $p = 1$ and $p = 2$ are shown in Figure 5-22. These are the background meshes prior to cutting. As such, tetrahedra outside the computational domain

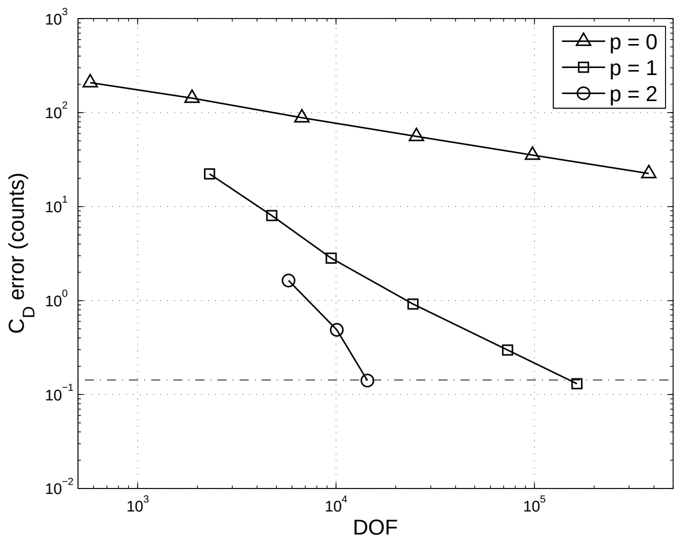
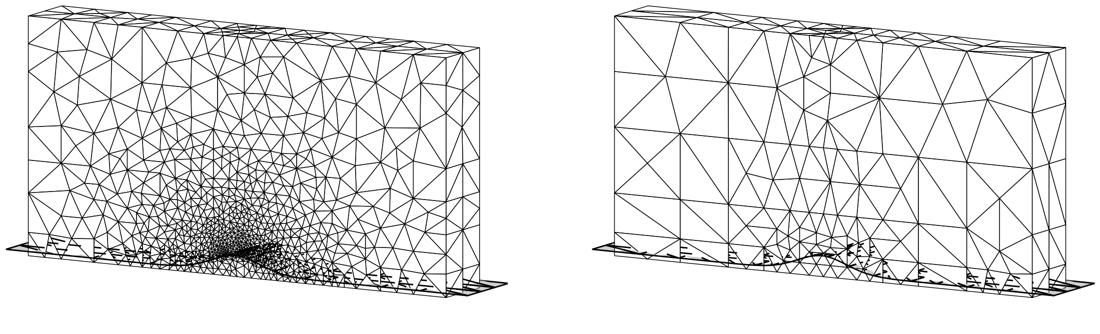


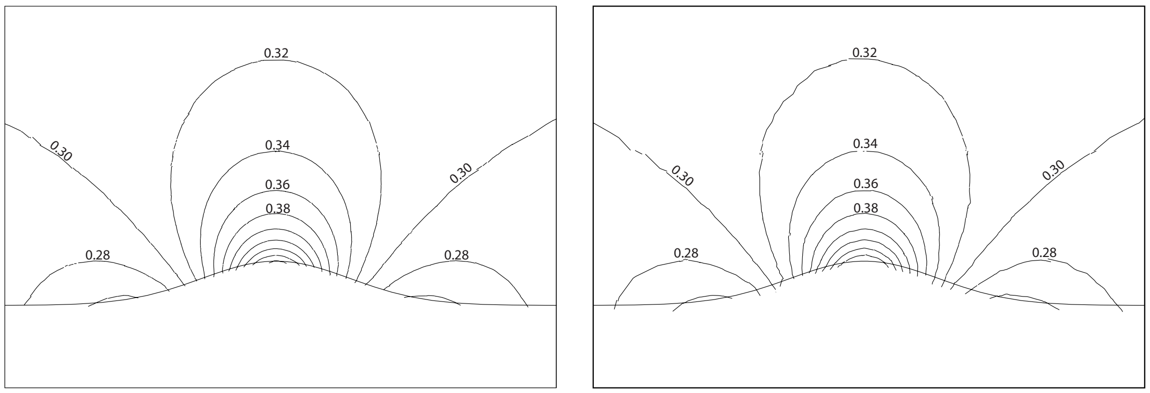
Figure 5-21: Gaussian bump channel: $M = 0.3$. Drag output error vs degrees of freedom. Dashed line indicates prescribed tolerance of $e_0 = 0.15$ counts.



(a) $p = 1$: 51264 elements (b) $p = 2$: 1787 elements

Figure 5-22: Gaussian bump channel: $M = 0.3$. Final adapted meshes for $p = 1$ and $p = 2$.

are still shown. As expected, refinement is concentrated in the vicinity of the bump, inside the computational domain. The $p = 2$ mesh is much coarser than the $p = 1$ mesh, requiring only 1787 elements as opposed to 51264 elements for $p = 1$. For comparison, the finest $p = 0$ mesh contains 377042 elements and yields an error of over 20 counts. Mach number contours for solutions on the adapted $p = 1$ and $p = 2$ meshes are shown in Figure 5-23. The plots are shown for a planar cut parallel to the x–z plane taken down the middle of the channel. Since the solution is on a cut-cell mesh, some of the contours extend through the geometry. This is a by-product of the visualization, as the solution is rendered on tetrahedra of the background mesh. For the flow solver, the solution is only physically valid inside the computational domain. The contours are similar for $p = 1$ and $p = 2$. The $p = 2$ contours appear slightly more rugged, although this is in part due to the fact that the $p = 2$ solution is plotted using linear rendering on a uniformly-refined mesh.



(a) $p = 1$ (b) $p = 2$

Figure 5-23: Gaussian bump channel: $M = 0.3$. Mach number contours on the final adapted meshes for $p = 1$ and $p = 2$. The planar cut is parallel to the $x{-}z$ plane and is taken down the middle of the channel.

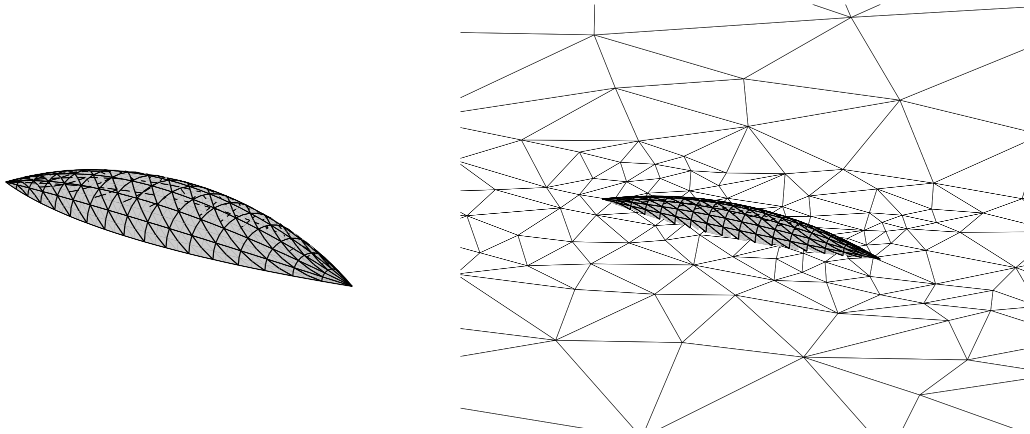
5.2.2 $M = 0.3$ Flow Over a Body of Revolution

This case considers inviscid flow over a body of revolution. The geometry surface is described analytically by

$$r = 0.3x(1 - x), \quad y = r\cos(\theta), \quad z = r\sin(\theta), \qquad 0 \leq x \leq 1, \quad 0 \leq \theta \leq \pi.$$

$M = 0.3$ freestream flow is imposed in the positive x direction, leading to an axi-symmetric, three-dimensional flow around the body of revolution. For the adaptive runs, drag serves as the output of interest. The true value of the drag, again nonzero due to proximity of the boundaries, is computed using a $p = 2$ solution on an adapted cut-cell mesh with over 60000 elements.

By symmetry, the simulation of flow around any finite wedge, $\Delta\theta$, of the body of revolution is sufficient. For this case, half of the body of revolution, $\Delta\theta = 180^o$, was used. A quadratic patch representation of the half body of revolution is shown in Figure 5-24a. 256 patches are used to represent the surface, with 8 divisions in θ and 16 divisions in x. An



(a) 256 surface patches

(b) 2883-element initial mesh

Figure 5-24: Body of revolution: $M = 0.3$. Quadratic patch representation of the geometry (a) and the initial background mesh (b).

initial background mesh for adaptation was generated via TetGen by adapting a uniform mesh of the background domain box to the geometry. The farfield position of the box was located roughly five chord lengths from the body of revolution. A plot of the mesh on the symmetry ($z = 0$) plane is shown in Figure 5-24b. As in the Gaussian-bump channel case, the embedded geometry extends out of the background domain to prevent degeneracies during the cutting.

Adaptive runs were performed for $p = 0, 1, 2$, using 1 count for the drag tolerance. The drag coefficient was computed using the frontal cross-section as the reference area. Figure 5-25 shows the results of the runs. As in the Gaussian bump case, adaptation for $p = 0$ was not continued down to the drag tolerance because computational costs became prohibitive. Comparing $p = 2$ and $p = 1$ at the error tolerance, $p = 2$ requires about a factor of 8 fewer degrees of freedom than $p = 1$. In terms of the estimated work, $p = 2$ is therefore about 3 times less expensive than $p = 1$ and at least several orders of magnitude less expensive than $p = 0$.

Symmetry-plane plots of the adapted meshes for $p = 1$ and $p = 2$ are shown in Figure 5-26. Compared to the final $p = 1$ mesh, the final $p = 2$ mesh achieves a lower error level with about a factor of twenty fewer elements. As expected, both meshes are refined primarily at the nose and tail of the body of revolution. Mach number contours for the solutions on these final adapted meshes are shown in Figure 5-27. The planar cut for these contours was taken slightly above the symmetry plane, at $z = .03$. The solutions are similar for $p = 1$ and $p = 2$. In both cases, the contours are smooth and no dissipation wake is observed.

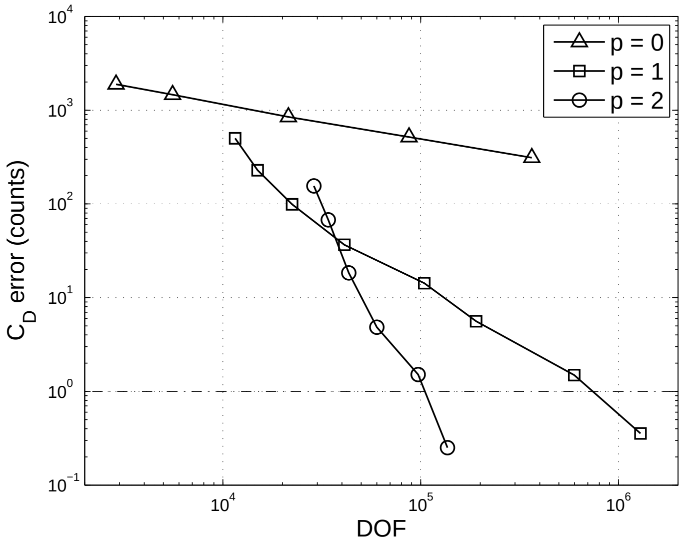
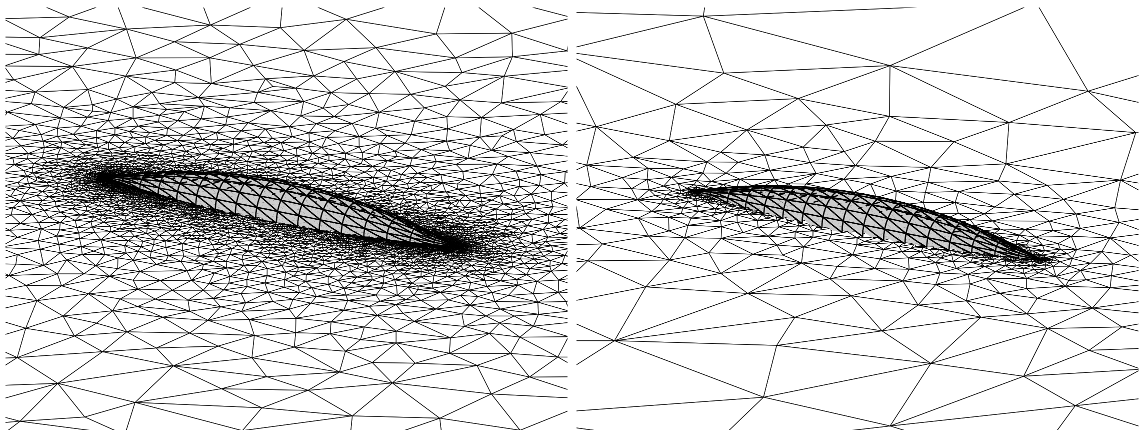


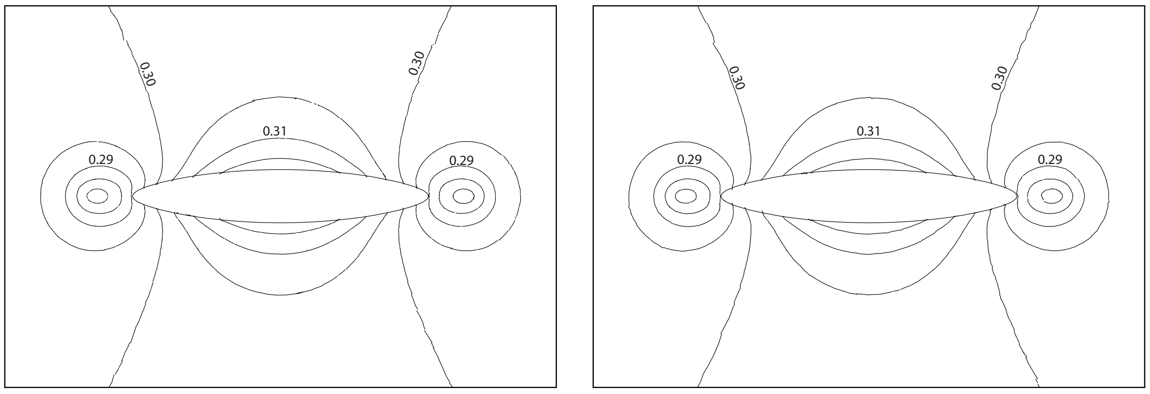
Figure 5-25: Body of revolution: $M = 0.3$. Drag output error versus degrees of freedom. Dashed line indicates prescribed tolerance of $e_0 = 1$ count.



(a) $p = 1$: 322894 elements

(b) $p = 2$: 13669 elements

Figure 5-26: Body of revolution: $M = 0.3$. Final adapted meshes for $p = 1$ and $p = 2$.



(a) $p = 1$ (b) $p = 2$

Figure 5-27: Body of revolution: $M = 0.3$. Mach number contours on the final meshes for $p = 1$ and $p = 2$. The cut is parallel to the $x-y$ plane at a height $z = .03$.

5.2.3 $M = 0.1$, $\alpha = 0^o$ Flow Over a NACA 0012, $AR = 2$ Wing

In this case, inviscid flow is modeled over a rectangular, untwisted wing with aspect ratio (AR) of 2 and with a constant, NACA 0012 airfoil section. At the wing tip, the symmetric airfoil is rotated to form half a body of revolution, effectively removing any sharp corners from the geometry. Since the wing is symmetric about the root, only half of the wing is modeled. For the adaptive runs, drag is used as the output of interest. This drag is nonzero due to the proximity of the farfield, which consists of a rectangular box about five chord lengths away from the wing. The true value for this drag is computed from a $p = 2$ solution on an adapted cut-cell mesh with over 60000 elements.

Figure 5-28a shows the quadratic patch representation of the wing geometry. 868 patches are used, spaced evenly along the span, and with curvature-based spacing along the chord to sufficiently resolve the leading edge. An initial background mesh was constructed using TetGen by adapting a uniform mesh of the farfield box to the geometry. The resulting 4419-element mesh is depicted along the symmetry plane in Figure 5-28b. As in the previous cases, the surface geometry extends slightly out of the background domain to prevent cutting degeneracies.

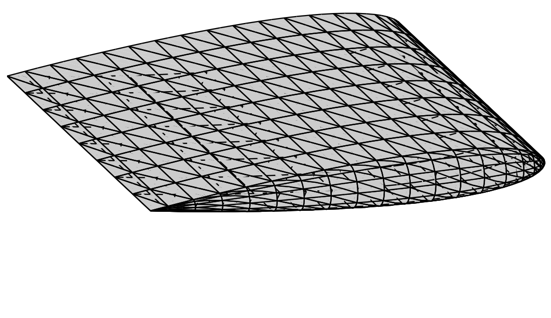
Adaptive runs were performed for $p = 0, 1, 2$ using 2 drag counts for the tolerance. The drag coefficient was computed using the wing planform area. Figure 5-29 shows the convergence of the output error versus degrees of freedom. Again, convergence with $p = 0$ is very slow, and both the degree of freedom count and the computational work can be estimated to be several orders of magnitude greater than for $p = 1$ and $p = 2$. Both $p = 1$

and $p = 2$ achieve the error tolerance, at which point $p = 1$ requires about 3.5 more DOF. In terms of estimated work at the error tolerance, $p = 2$ is only slightly advantageous over $p = 1$, although the steeper slope for $p = 2$ suggests that it will become increasingly more favorable for stricter error tolerances.

The symmetry-plane plots of the final adapted meshes for $p = 1$ and $p = 2$ are shown in Figure 5-30. Noticeable in the $p = 1$ mesh is the high refinement at the leading and trailing edges. For comparison, the finest $p = 0$ mesh contains 457229 elements and achieves an error of about 100 counts. Mach number contours for the solutions on these final adapted meshes are shown in Figure 5-31. The planar cut for the figure is taken at 60% of the half-span of the wing. The $p = 1$ and $p = 2$ solutions appear similar: the contours are slightly rugged, and a slight wake is present due to numerical dissipation.

5.2.4 $M = 0.1$, $\alpha = 0^o$ Flow Over a Wing-body Configuration

The previous results demonstrated the performance of 3D cut-cells and output-based adaptation for several curved geometries. Also of interest, however, is the robustness of the cut-cell, adaptive method for practical problems of interest. To verify this robustness, cut-cell adaptation was applied to the wing-body geometry shown in Figure 5-32. This geometry was used in the third Drag Prediction Workshop. The geometry is represented with 9368 quadratic patches that are spaced to minimize geometry interpolation error in areas of high curvature.


(a) 868 surface patches

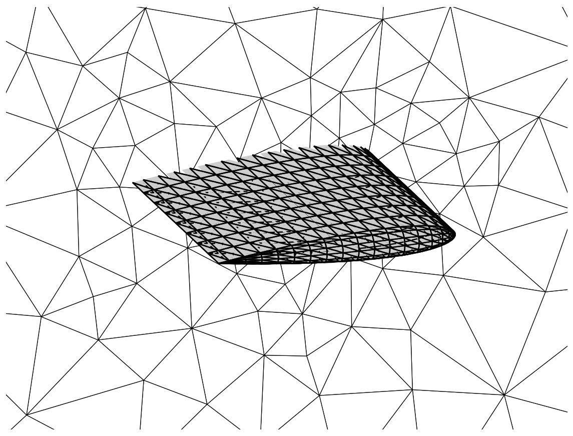

(b) 4419-element initial mesh

Figure 5-28: NACA 0012 wing: AR=2, $M = 0.1, \alpha = 0^o$. Quadratic patch representation of the geometry (a) and the initial background mesh (b).

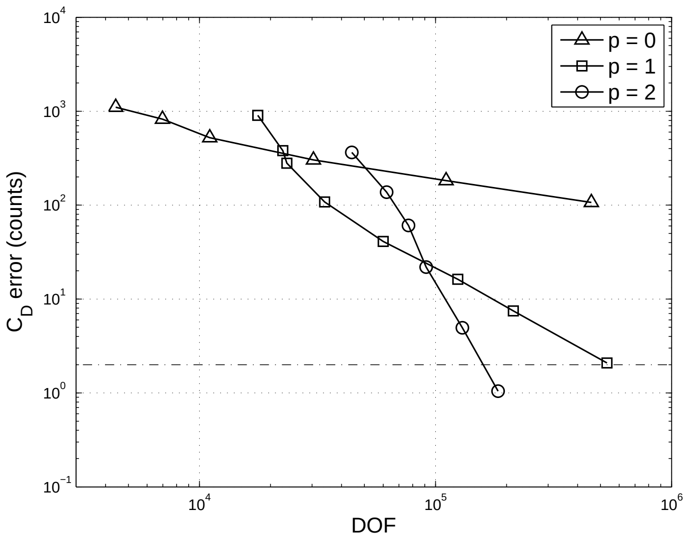
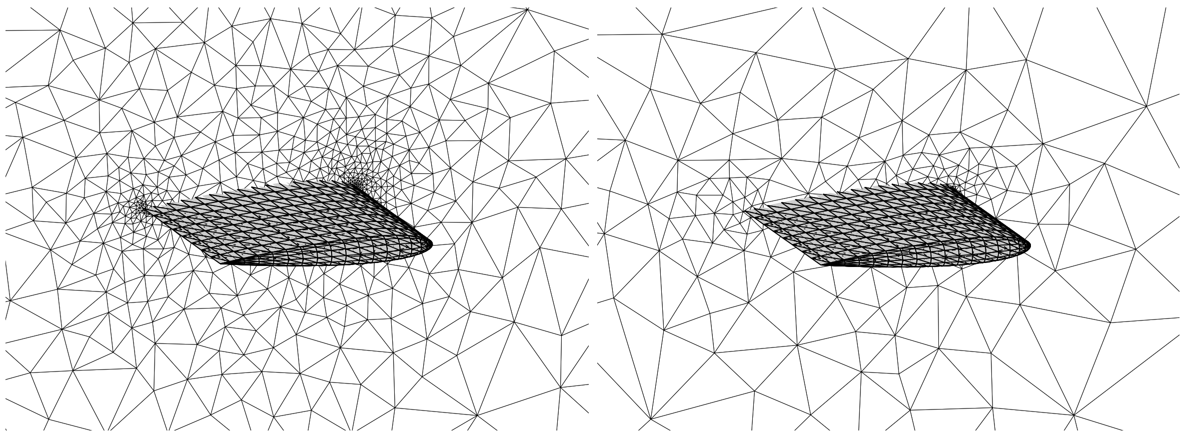
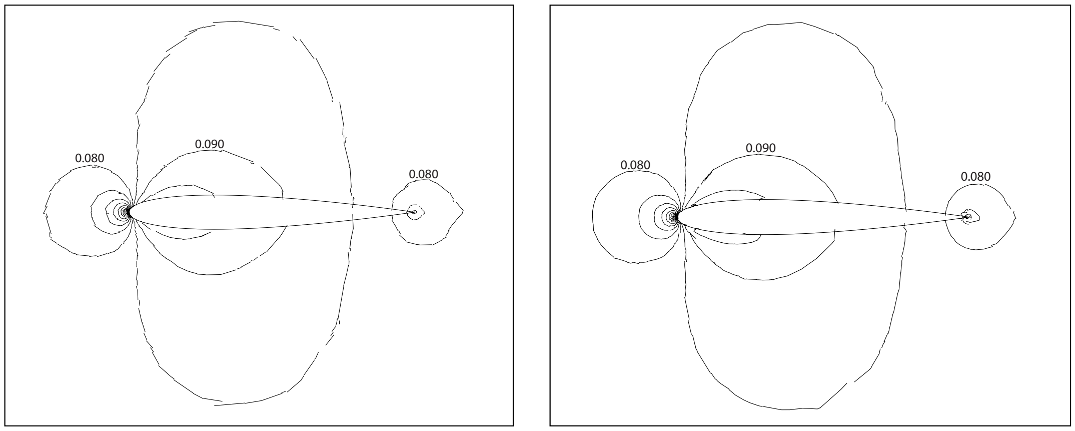


Figure 5-29: NACA 0012 wing: AR=2, $M = 0.1, \alpha = 0^o$. Drag output error versus degrees of freedom. Dashed line indicates prescribed tolerance of $e_0 = 2$ counts.



(a) $p = 1$: 133110 elements (b) $p = 2$: 18407 elements

Figure 5-30: NACA 0012 wing: AR=2, $M = 0.1, \alpha = 0^o$. Final adapted meshes for $p = 1$ and $p = 2$.



(a) $p = 1$ (b) $p = 2$

Figure 5-31: NACA 0012 wing: AR=2, $M = 0.1, \alpha = 0^o$. Mach number contours on the final meshes for $p = 1$ and $p = 2$. The cut is parallel to the $x-z$ plane and is situated at 60% of the half-span.

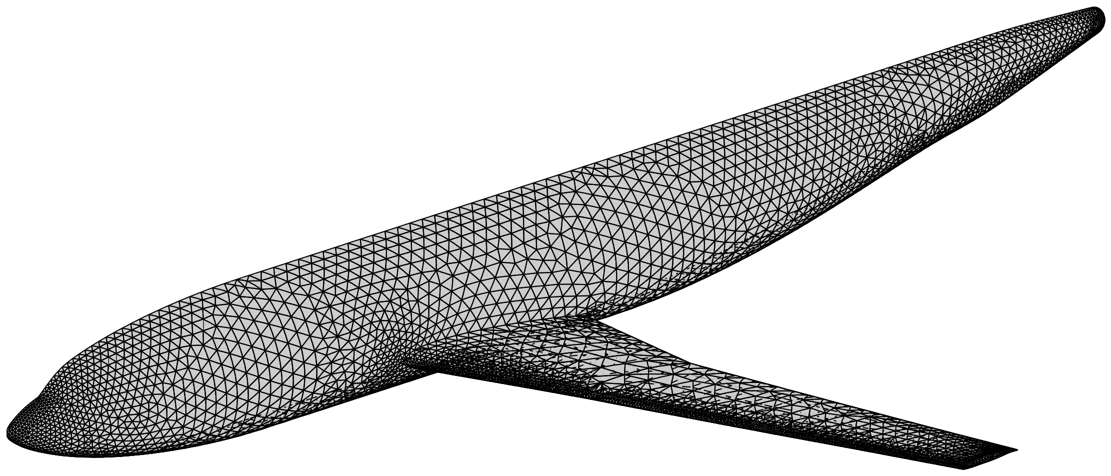


Figure 5-32: Wing-body: $M = 0.1, \alpha = 0^o$. Surface representation with 9368 quadratic patches.

A coarse, geometry-adapted mesh of 20447 elements served as the initial mesh for adaptive runs at orders $p = 0, 1, 2$. Adaptation was based on drag, with 1 error count as the tolerance. The wing planform area was used to non-dimensionalize the drag. Figure 5-33 shows the convergence of the drag coefficient for the three orders. For this geometry, no "true" drag value was available due to computational limitations. Nevertheless, Figure 5-33

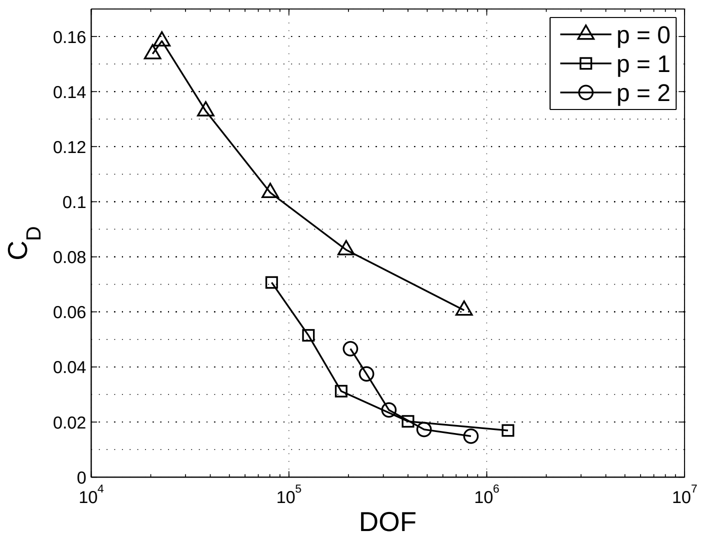
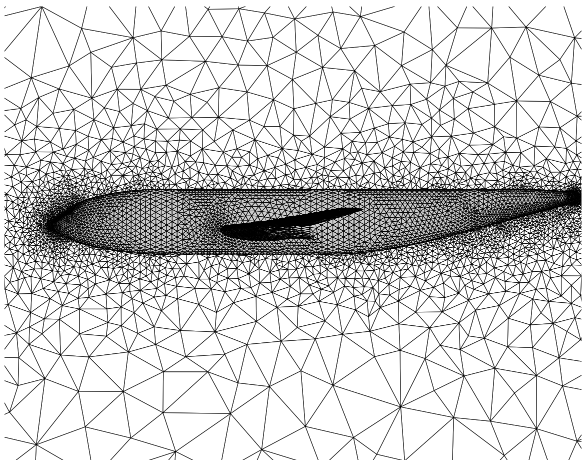


Figure 5-33: Wing-body: $M = 0.1, \alpha = 0^o$. Drag output versus degrees of freedom.

indicates that $p = 0$ is converging much more slowly compared to $p = 1$ and $p = 2$. After the initial adaptation iterations, $p = 2$ appears to be converging more quickly than $p = 1$. More importantly, the successful application of cut cells and adaptation in this case demonstrates the robustness and automation possible for a practical geometry configuration.

Figure 5-34 compares the finest $p = 1$ and $p = 2$ meshes. The meshes are plotted on the symmetry plane, and the quadratic-patch surface is overlaid. As expected, areas of refinement on the symmetry plane include the nose and the tail. Away from the symmetry plane, the leading and trailing edges of the wing also exhibit high refinement. Finally, Figure 5-35 shows the Mach number contours for the finest $p = 1$ and $p = 2$ mesh solutions, at a section of the wing 50% along the half-span. Both contour plots exhibit a slight dissipation wake off the trailing edge, indicating that the flow is not very highly resolved. The $p = 1$ and $p = 2$ contours are very similar, with $p = 2$ exhibiting slightly smoother features compared to $p = 1$.


(a) $p = 1$: 320245 elements

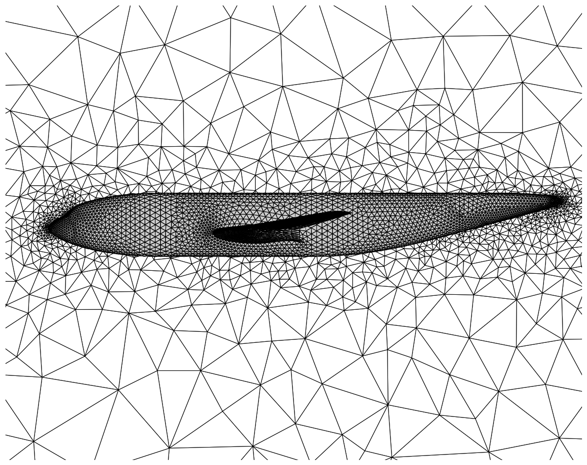
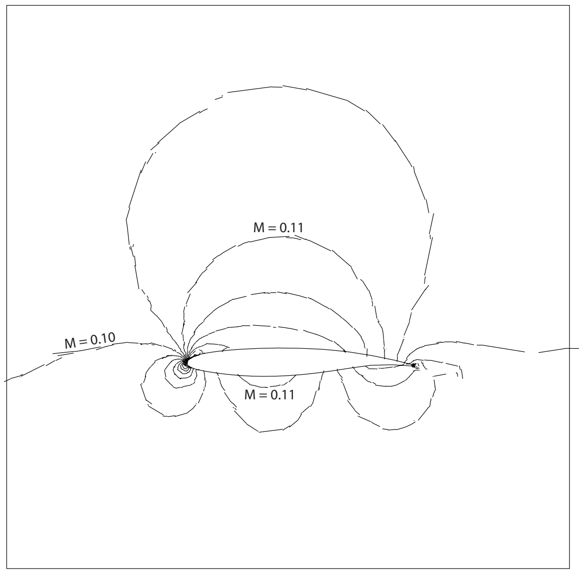
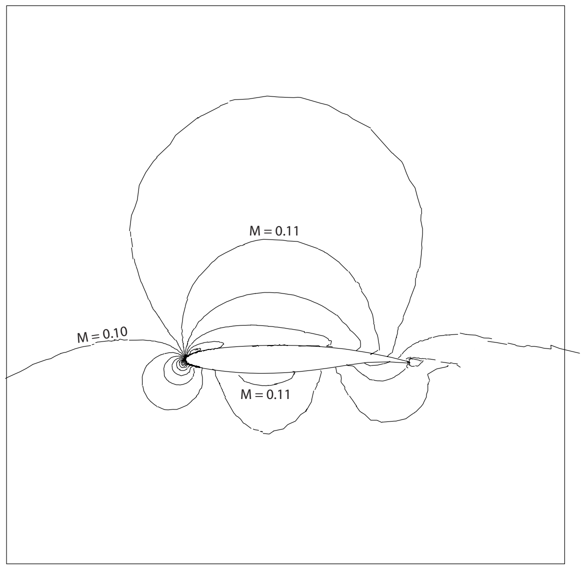

(b) $p = 2$: 83193 elements

Figure 5-34: Wing-body: $M = 0.1, \alpha = 0^o$. Finest adapted meshes for $p = 1$ and $p = 2$.



(a) $p = 1$



(b) $p = 2$

Figure 5-35: Wing-body: $M = 0.1, \alpha = 0^o$. Mach number contours on the finest meshes for $p = 1$ and $p = 2$. The cut is parallel to the $x - z$ plane and is situated at 50% of the half-span.

Chapter 6

Conclusions and Future Work

6.1 Summary

This thesis presents a complete output-based mesh adaptation procedure for high-order discontinuous Galerkin discretizations in two and three dimensions. The key components of this method are anisotropic, output-based mesh adaptation for high-order solutions and simplex, cut-cell meshing. Together, these ideas address two shortcomings in current CFD practices: insufficient automation and insufficient robustness in the geometry-to-solution process. Output-based mesh adaptation allows for an automated, goal-oriented solution method in which an output error is driven to a prescribed user tolerance. The accompanying error estimate additionally provides engineering confidence in the final results. As the adaptive process involves changing the computational mesh, automation and robustness of the mesher are important ingredients. To this end, a simplex, cut-cell meshing technique is introduced as an alternative to standard, boundary-conforming mesh generation, which currently lacks sufficient robustness and automation for complex, curved geometries. The cut-cell meshing technique is similar to existing Cartesian cut-cell techniques, except that simplices, triangles in two dimensions and tetrahedra in three dimensions, are used instead of lattice-bound rectangles or hexahedra. These simplices permit anisotropic stretching in general directions to efficiently resolve thin boundary and shear layers. The fact that the cut-cell meshes do not have to conform to the geometry boundary is an enabling feature that greatly improves the automation and robustness of mesh generation.

Output error estimation is performed by solving a linear adjoint problem associated with the output of interest. The adjoint serves as a Green's function relating local residuals to errors in the output, properly accounting for propagation effects inherent to hyperbolic problems. Together with estimates of the interpolation error, which are obtained by H_1 patch-reconstruction of the solution, the adjoint yields an estimate of the output error. This estimate is not a bound, although its validity improves as the discrete solution approaches the exact solution. Localized to the elements, the output error serves as an indicator for mesh adaptation. In isotropic adaptation, this indicator is translated directly into a requested mesh size for each element. In anisotropic adaptation, the indicator provides an absolute mesh size that is used in combination with relative sizing obtained from directional interpolation error estimates. A high-order extension of the Hessian-matrix approach is proposed for calculating these relative sizes. The final mesh size request for each element is the result of an optimization process, the objective of which is to meet the user-specified error tolerance while equidistributing the error over the elements. For the purposes of re-meshing, the mesh size request is expressed as a Riemannian metric for each element.

Simplex cut-cell meshes are produced by intersecting an embedded geometry with a volume-filling background mesh. The embedded geometry is represented by cubic splines in 2D and quadratic patches in 3D. The 2D cutting algorithm involves solving cubic equations, while the 3D cutting algorithm requires intersections between conic sections. Since the resulting cut cells and cut faces may be irregular in shape, a technique is proposed for integrating functions over arbitrarily-shaped areas and volumes. This technique is based on projecting the integrand onto a high-order tensor product space using least-squares minimization with randomly chosen sampling points inside the cut cell. The basis for the tensor product space is constructed by taking the divergence of appropriately-chosen vector-valued functions of one degree higher than the desired order of integration. By the divergence theorem, integrals of the basis functions over the cut cells can be expressed as integrals over the cut-cell boundaries. For 2D cut cells, the 1D boundary integrals are evaluated using Gauss quadrature. For 3D cut cells, a similar approach is first used to derive integration rules on the irregular cut and embedded faces, which are then used in deriving the volume integration rules.

The integration rules over cut cells and cut faces consist of sampling points and associated weights that are derived in a pre-processing step before solution iteration. These rules are reconstructed every time the grid changes, which currently occurs at every adaptation iteration. Metric-driven meshing of the computational domain is performed using existing mesh generation packages. With the boundary-conforming constraint removed, mesh generation is simple and robust. Thus, given a geometry, an initial coarse mesh of the background domain, and a requested output error tolerance, the adaptive solution process is fully automated.

6.2 Conclusions

From the two-dimensional results, several conclusions can be drawn about the performance of the adaptation algorithm. First, the output-based error estimate, while not a bound on the error, successfully drives the adaptation for nearly all of the cases tested to produce solutions that meet the prescribed error tolerance on the output. This conclusion is qualified by the fact that several of the cases required a reasonable starting mesh to produce an accurate initial error estimate. Second, based on adaptation runs using a variety of initial meshes, the final meshes appear relatively insensitive to the starting meshes, given a sufficiently-low error tolerance. Third, adaptation on $p = 2$ and $p = 3$ is observed to produce final meshes that more efficiently use degrees of freedom compared to $p = 1$ meshes. The difference in degrees of freedom is especially significant for the smooth, inviscid test case and for the high Reynolds number and high Peclet number boundary-layer cases. The advantage of $p = 3$ and $p = 2$ holds even when accounting, in an approximate fashion, for the additional computational work per degree of freedom required for high-order solutions. Fourth, the cut-cell method produces results similar to those obtained with boundary conforming meshes. Moreover, the cut-cell method is observed to be robust, even for very highly-anisotropic boundary-layer meshes.

In the three-dimensional results, comparisons to boundary-conforming adaptive results are not presented due to the present unavailability of robust, three-dimensional meshing. Nevertheless, accuracy of the cut-cell method is verified for the channel-flow case by comparing to a boundary-conforming result on a manually-generated mesh. The cut-cell adaptive runs illustrate that, for the error tolerances tested, efficiency in use of degrees of freedom improves with increasing interpolation order, at least up to $p = 2$. As in two-dimensions, this advantage also holds for the approximate work estimate. More importantly, the fact that the cut-cell meshing and adaptation remain robust for arbitrary, three-dimensional geometries demonstrates the applicability of cut cells to practical simulations. Of course, as much of this work is for proof-of-concept, the techniques used exhibit certain shortcomings and inefficiencies that are addressed in the following section in the context of ideas for future work.

6.3 Future Work

During the course of this work, several ideas for potential future work were identified. These ideas, listed below, range from improvements or extensions of current methods to new directions.

1. **Improved sampling point selection for cut cells.**

   Sampling points for integration on cut cells are currently chosen randomly via ray-casting from enclosing boundaries. While oversampling is used to mitigate effects of randomly-occurring clusters, ill-conditioned sets of points are still possible. Furthermore, oversampling increases the computational cost of integration on cut cells and cut faces. A more sophisticated sampling point selection process, for example, based on an electric-charge analogy, may significantly reduce the required number of sampling points. In addition, elements cut in a prototypical fashion may be mapped to a set of reference elements with pre-computed optimal quadrature rules, further reducing the computational cost.

2. **Alternate geometry representation in three dimensions.**

   Although quadratic patches are reasonably efficient at minimizing the interpolation and slope errors in the geometry representation, they do not exactly model general CAD geometries, and they still permit finite slope discontinuities between the patches. If sufficiently resolved, these slope discontinuities can lead to solution singularities. In an adaptive setting, an alternate option to using a static patch representation is to employ a callback to the true CAD geometry to re-tessellate the surface at every adaptation iteration using the background mesh sizing as an indicator of the required refinement level. Another idea is to employ higher-order patches or an exact geometry representation, although the cutting algorithm would likely be more complex.

3. **More general adaptive criteria.**

   Currently, an adjoint-based error estimate for a single output is translated into an absolute mesh size magnitude via *a priori* output error convergence predictions while interpolation errors in the Mach number yield the directional relative sizes. Choosing the Mach number for anisotropy detection works well in practice, but has little rigorous foundation. More general adaptation could conceivably incorporate anisotropy directly into the output error estimates, as done for certain equation sets by Formaggia *et al* [26]. In addition, multiple outputs could be incorporated into the error estimation, metric definition, and mesh optimization, as done by Hartmann and Houston [35], with the goal of obtaining a single, multi-purpose solution.

4. **Use of curved elements in the background mesh.**

   A thin boundary layer on a curved geometry exhibits strong variation normal to the boundary, but relatively little variation in the curved, streamwise direction. Linear, as opposed to curved (i.e. $q > 1$), simplex elements are relatively inefficient at resolving such features. As such, in the presence of curvature, a large number of elements is required in the streamwise direction to maintain adequate resolution in the normal direction. This issue is identical to that of using linear patches to represent a curved geometry. As discussed in Appendix F, quadratic patches can dramatically reduce the patch count for a desired low geometry interpolation error. Similarly, $q = 2$ or higher-order curved elements can significantly reduce the element count for thin boundary layers. However, issues that need to be addressed for this method to be successful include robust background mesh generation and accurate geometry intersection with curved elements.

5. **Development of an *hp*-adaptive algorithm.**

   The adaptation strategy used in this work consists of h-adaptation at a constant order, p. As discussed in Section 3.2, a more efficient strategy is one that incorporates p-adaptation as well. The mesh optimization algorithm in this case would need to choose between h and p adaptation for each element via some form of regularity estimation. While such an *hp*-adaptive algorithm would certainly be more complex, the potential computational savings in more efficient use of degrees of freedom could be significant.

6. **Extension to other equation sets.**

   While the target application in this work is aerodynamics, the adaptation method and the cut-cell algorithm are readily extendable to different equation sets. Specifically, the adaptive method only requires an output with a well-posed adjoint problem, while the cut-cell method offers potential robustness and automation advantages for any computations on complex geometries, especially in the presence of highly-anisotropic boundary-layer features. Of particular interest is the extension of the cut-cell method to unsteady simulations with grid motion; a challenging task in which efficiency of the cutting algorithm would be of paramount importance.

Appendix A

Compressible Navier-Stokes Boundary Conditions

This appendix describes how boundary conditions are imposed for the compressible Navier-Stokes equations. This description is taken primarily from [58], and is included here for completeness.

Full State

In this case, all K components of the boundary state vector, $\mathbf{u}_H^b$, are specified. The inviscid flux is computed using the Roe-averaged flux function, $\widehat{F}_{ki}(\mathbf{u}_H^+, \mathbf{u}_H^b)$. The viscous fluxes are computed as indicated in Table 2.1 for the Dirichlet case.

Inflow/Outflow

For an inflow or outflow, the boundary state vector, $\mathbf{u}_H^b$, is constructed from the outgoing Riemann invariants and from the specified boundary data. Specifically, for a subsonic inflow, the total temperature, total pressure, and flow direction are prescribed, whereas the J^+ Riemann invariant is taken from the interior state. For a subsonic outflow, the static pressure is prescribed, whereas the J^+ Riemann invariant, the entropy, and the tangential velocity are taken from the interior state.

The inviscid flux is then computed using the analytical flux evaluated with $\mathbf{u}_H^b$: $F_{ki}(\mathbf{u}_H^b)$. Note, the Roe-averaged flux is not used in this case. The viscous fluxes are evaluated as in Table 2.1 for the Dirichlet case, using the constructed boundary state $\mathbf{u}_H^b$.

Adiabatic No-Slip Wall

For an adiabatic no-slip wall, the boundary state, $\mathbf{u}_H^b$, is constructed using density and energy from the interior state and with the d velocities set to zero. The inviscid flux is computed using the analytical flux evaluated with $\mathbf{u}_H^b$: $F_{ki}(\mathbf{u}_H^b)$. Again, the Roe-averaged flux is not used. The viscous fluxes are evaluated based on the Dirichlet case in Table 2.1, with the exception that the the viscous energy flux, $\widehat{Q}_{(d+2)i}n_i$, is set to zero to satisfy the adiabatic wall condition.

Appendix B

Adapting for $p > 1$ Interpolation

The following example shows the problem of using the Hessian matrix for measuring high-order anisotropy. Consider the following function $u(x, y)$ on a unit square domain, $[0,1] \times [0,1]$:

$$u = 1.0 + (x^2 + 16y^2) + \epsilon(64x^3 + y^3), \tag{B.1}$$

where $\epsilon << 1$. The Hessian matrix is

$$H = \begin{bmatrix} 2 & 0 \\ 0 & 32 \end{bmatrix} + O(\epsilon). \tag{B.2}$$

Ignoring $O(\epsilon)$ terms, the required element aspect ratio is calculated to be

$$AR \equiv \frac{\Delta x}{\Delta y} = \sqrt{\frac{32}{2}} = 4.0.$$

To test how interpolation accuracy varies with grid AR, a sequence of three grids, $AR = 0.25$, $AR = 1.0$, and $AR = 4.0$ was constructed, each with 32 elements (Figure B-1).

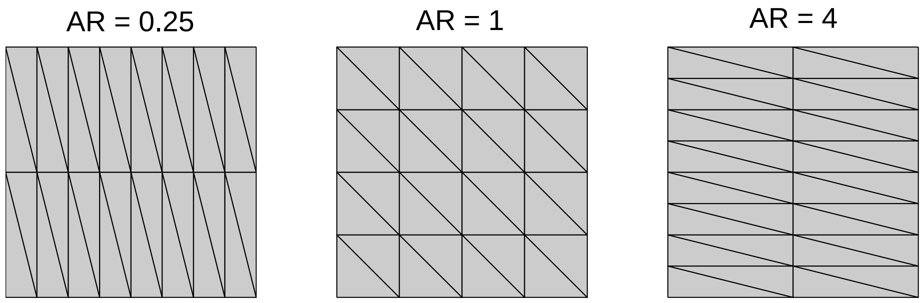


Figure B-1: Three grids used for the interpolation of the function u in (B.1). Hessian-based analysis predicts $AR = 4.0$ as the ideal for interpolation.

Table B.1: L_2 norm of interpolation error on each grid for $p = 1$ and $p = 2$.

	$p = 1$	$p = 2$
$AR = 0.25$	2.31E-1	2.19E-7
$AR = 1.0$	5.91E-2	1.42E-6
$AR = 4.0$	2.37E-2	1.14E-5

The L_2 norm of the interpolation error on each grid was calculated for $p = 1$ and $p = 2$ interpolation. $\epsilon = 10^{-4}$ was used for the calculations. Table B.1 shows the results. As expected from the Hessian matrix analysis, for $p = 1$, the grid with $AR = 4.0$ shows the smallest interpolation error. For $p = 2$, however, $AR = 0.25$ produces the smallest interpolation error per degree of freedom, out of the three cases tested. $AR = 4.0$ produces an error 50 times larger. Thus, blindly using the results from the Hessian matrix for high-order interpolation can produce a clearly sub-optimal grid.

Considering the third order derivatives in (B.1) gives insight into the AR expected for $p = 2$ interpolation. Specifically, $u_{xxx} = 384\epsilon$ and $u_{yyy} = 6\epsilon$. For this problem, these values happen to be the maximum and minimum third-order derivatives, and they also happen to occur in orthogonal directions (x and y). Thus, elements that equidistribute the interpolation error in these directions should have

$$u_{xxx}(\Delta x)^3 = u_{yyy}(\Delta y)^3 \quad \Rightarrow \quad AR = \frac{\Delta x}{\Delta y} = \left(\frac{u_{yyy}}{u_{xxx}}\right)^{1/3} = \left(\frac{6\epsilon}{384\epsilon}\right)^{1/3} = 0.25.$$

This is the AR that produced the smallest $p = 2$ interpolation error out of the three cases tested (c.f. Table B.1).

Appendix C

Refinement Prediction Example

One drawback of Zienkiewich and Zhu's standard refinement prediction method lies in its error equidistribution capability. In particular, the method may lead to over-refinement of elements with large error indicators and under-refinement of elements with small error indicators. This problem occurs due to the use of error equidistribution on the current mesh rather than on some reasonable prediction of the final mesh.

To illustrate the significance of this problem, consider a two-element, one-dimensional mesh, with error indicators $\epsilon_1 = 1$ and $\epsilon_2 = 16$, and a global error target of $e_0 = 1.0$. Assume that the error indicators are additive (e.g. L_1 norm of error), and that the *a priori* error estimate is $\epsilon_k \sim h_k^1$.

Using the refinement prediction method with two elements ($N = 2$), the permissible error on each element is $\bar{e}_0 = e_0/N = 1.0/2 = 0.5$. Calculation of the new h_k, via (3.19), leads to element 1 being split into 2 children and element 2 being split into 32 children, for a total of 34 elements in the new grid (lower left in Figure C-1). Assuming the *a priori* estimate is exact, the global error target is met, since each of the areas covered by the original two elements contributes 0.5 to the global error. However, the error is not equidistributed on the new grid. Each child of element 1 has an error of $0.5/2 = 0.25$, while each child of element 2 has an error of $0.5/32 = 0.015625$. Element 2 has been over-refined while element 1 has not been refined enough to satisfy error equidistribution on the resulting mesh.

Now consider an alternate refinement in which element 1 is split into 5 elements while element 2 is split into 20 elements, for a total of 25 elements in the new mesh (lower right in Figure C-1). Altogether, the children of element 1 now contribute 1/5 to the global error, so each contributes $1/5^2 = 1/25$. Also, the children of element 2 contribute a cumulative $16/20 = 4/5$ to the global error, so each contributes $(4/5)/20 = 1/25$. The global error target is met ($1/5 + 4/5 = 1.0$), and the error is equidistributed on the final

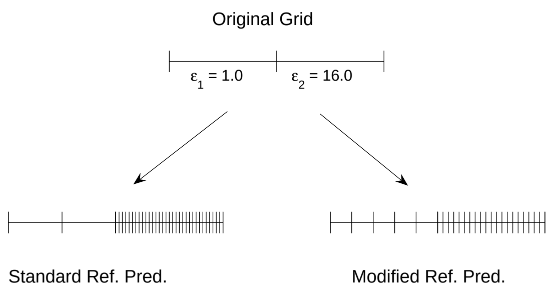


Figure C-1: Example of global mesh modification on a 1D mesh using standard refinement prediction (lower left) and using an alternate goal-oriented error equidistribution method (lower right).

mesh. Finally, the number of elements in this mesh is fewer than the 34 obtained from refinement prediction.

The optimal mesh can be deduced systematically by solving the equations representing the global error requirement and error equidistribution on the adapted mesh. In the example presented, assume elements 1 and 2 are divided into n_1 and n_2 elements, respectively. To satisfy the global error requirement,

$$\frac{1}{n_1} + \frac{16}{n_2} = e_0 = 1. \tag{C.1}$$

To satisfy error equidistribution *among the children elements*,

$$\frac{1}{n_1}\frac{1}{n_1} = \frac{1}{n_2}\frac{16}{n_2}. \tag{C.2}$$

The *a priori* error estimate was used in deriving both of these equations. Solving these two equations for n_1 and n_2 yields $n_1 = 5$ and $n_2 = 20$. This is an example of the discrete systematic approach discussed in Section 3.2.2.

Appendix D

Cubic Spline Intersection

The intersections between a cubic spline segment and an ordinary line can be found analytically. Consider a spline segment between two knots (labeled 1 and 2), as shown in Figure D-1.

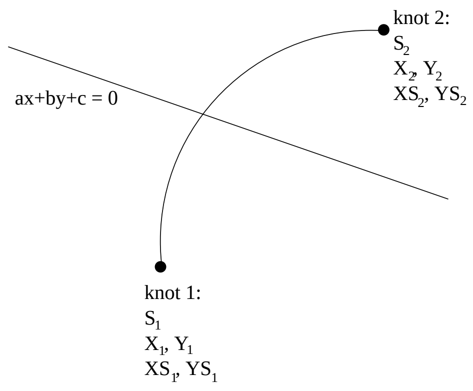


Figure D-1: Intersection between a spline segment and a line. A root-finding formula is used to determine where a spline segment cuts an interior face.

The spline information consists of, at each knot, the knot coordinates and the derivatives of each coordinate with respect to the arc-length parameter, s. In practice, these derivatives are computed by a splining algorithm, given a set of knot coordinates. The quantities associated with knot 1 are $S_1, X_1, Y_1, XS_1, YS_1$, where $XS \equiv dx/ds$, $YS \equiv dy/ds$, and similarly for knot 2. The (x, y) coordinates along the curved spline segment can be written as

$$\begin{aligned} x(t) &= tX_2 + (1-t)X_1 + (t-t^2)\big[(1-t)CX_1 - tCX_2\big], \qquad \text{(D.1)} \\ y(t) &= tY_2 + (1-t)Y_1 + (t-t^2)\big[(1-t)CY_1 - tCY_2\big], \end{aligned}$$

where $\Delta s = S_2 - S_1$, $t = (s - S_1)/\Delta s$, and

$$\begin{aligned} CX_1 &= XS_1\Delta s - (X_2 - X_1), \\ CX_2 &= XS_2\Delta s - (X_2 - X_1). \end{aligned}$$

CY_1 and CY_2 are defined similarly. The line segment is defined as those x, y pairs that satisfy $ax + by + c = 0$. Thus, an intersection must satisfy

$$\begin{aligned} 0 &= ax(t) + by(t) + c \\ &= \Big[a(CX_1 + CX_2) + b(CY_1 + CY_2)\Big]t^3 + \Big[a(-2CX_1 - CX_2) + b(-2CY_1 - CY_2)\Big]t^2 \\ &\quad + \Big[a(X_2 - X_1 + CX_1) + b(Y_2 - Y_1 + CY_1)\Big]t + \Big[aX_1 + bY_1 + c\Big] \\ &\equiv a_3t^3 + a_2t^2 + a_1t + a_0. \end{aligned} \tag{D.2}$$

Thus, an intersections is a solution to a cubic polynomial. Analytical formulas for cubic solutions exist. However, one must be careful of special cases and numerical conditioning. The cubic-root formula used in this work is given as follows. Consider a cubic equation in t with leading coefficient of 1,

$$t^3 + b_2t^2 + b_1t + b_0 = 0. \tag{D.3}$$

Define Q, R, and D by

$$Q \equiv \frac{3b_1 - b_2^2}{9}, \qquad R \equiv \frac{9b_2b_1 - 27b_0 - 2b_2^3}{54}, \qquad D \equiv Q^3 + R^2. \tag{D.4}$$

The number and type of roots are dictated by the value of D. In particular, the three possible cases $D = 0$, $D > 0$, and $D < 0$ are treated separately:

- $D = 0$: All roots are real, and at least two are the same:

$$\begin{aligned} t_0 &= -b_2/3 + 2R^{1/3}, \\ t_1 = t_2 &= -b_2/3 - R^{1/3}. \end{aligned}$$

  Note, a triple root occurs in this case when $R = 0$.

- $D > 0$: One real root:

$$t_0 = -b_2/3 + \left(R + \sqrt{D}\right)^{1/3}.$$

- $D < 0$: Note this implies $Q < 0$. Let $\theta \equiv \cos^{-1}\left[R/(-Q)^{3/2}\right]$. Three distinct roots:

$$\begin{aligned} t_0 &= 2\sqrt{-Q}\cos(\theta/3) - b_2/3, \\ t_1 &= 2\sqrt{-Q}\cos((\theta+2\pi)/3) - b_2/3, \\ t_3 &= 2\sqrt{-Q}\cos((\theta+4\pi)/3) - b_2/3. \end{aligned}$$

While the cubic formula is valid for a leading coefficient of 1, a_3 in (D.2) could be arbitrary. A straightforward division by a_3 may not be well-conditioned when a_3 is small, which is the case when a spline segment is close to linear. Ideally, the b_i coefficients should be neither much smaller nor much larger than unity to enable accurate calculation of Q, R, and D in (D.4). A transformation that has worked in practice is to solve for $r = a_3^{1/3}t$. Substituting for t in (D.2) yields

$$r^3 + a_2 a_3^{-2/3} r^2 + a_1 a_3^{-1/3} r + a_0 \quad = \quad 0.$$

t is then obtained as $t = a_3^{-1/3} r$. The special case of $|a_3|$ close to machine zero reduces the equation to a quadratic (at most), which is solved via the quadratic formula with a similar conditioning scaling.

Valid solutions consist of those t that are between 0 and 1 and for which the associated (x, y) lie within the line segment bounding box (intersections are generally sought for finite line segments, not for infinite lines). Double roots, or tangency intersections are not counted as intersections, while triple roots are. Node intersections near spline-segment endpoints have to be handled carefully, especially for near-tangency intersections. Specifically, if the cubic solution yields t close to 0 or 1, the equation residual, (D.2), is compared to the residual evaluated with $t = 0$ and $t = 1$. The value of t that yields the lowest residual is used.

Appendix E

Conic Parametrization

This appendix presents the parametrization of a conic in coordinates (X', Y') that have been rotated to eliminate the cross term, $X'Y'$. That is, the conic to be parametrized is given by

$$A'X'^2 + C'Y'^2 + D'X' + E'Y' + F' = 0. \tag{E.1}$$

For the parabola, ellipse, and hyperbola, a common parametrization will be used, of the form

$$r = \frac{l}{1 - e\cos(\theta - \theta_0)}, \tag{E.2}$$

where l, e, and θ_0 are specific to each conic.

Parabola

A horizontal parabola can be written as $2l(X' - X'_0) = (Y' - Y'_0)$, where

$$l = \frac{-D'}{2C'}, \quad X'_0 = \frac{E'^2}{4C'D'} - \frac{F}{D'}, \quad Y'_0 = -\frac{E'}{2C'}.$$

The focus of the parabola is given by $X'_f = X'_0 + l/2$, $Y'_f = Y'_0$, and the eccentricity, $e = 1$. If $l > 0$, θ_0 is set to 0; otherwise, l is set to $-l$ and θ_0 is set to π. These values can now be used to parametrize the parabola according to (E.2). A vertical parabola is handled in a similar fashion.

Ellipse

Degeneracies excluded, (5.7) can be re-written by completing squares as,

$$A''(X' - X'_0)^2 + C''(Y' - Y'_0)^2 = 1, \tag{E.3}$$

where

$$\begin{aligned} A'' &= \frac{-A'}{F''}, \quad C'' = \frac{-C'}{F''}, \quad F'' = F' - \frac{D'^2}{4A'} - \frac{E'^2}{4C'}, \\ X'_0 &= \frac{-D'}{2A'}, \quad Y'_0 = \frac{-E'}{2C'}. \end{aligned}$$

For an ellipse, $A'' > 0$ and $C'' > 0$. $a = \max\left(\sqrt{1/A''}, \sqrt{1/C''}\right)$ and $b = \min\left(\sqrt{1/A''}, \sqrt{1/C''}\right)$ are the semi-major and semi-minor axes of the ellipse, respectively. The eccentricity is $e = \sqrt{1 - b^2/a^2}$ and l for the parametrization is b^2/a. θ_0 is 0 if $\sqrt{1/A''} > \sqrt{1/C''}$ and π otherwise. The foci are located at $\pm ea$ along the semi-major axis from (X'_0, Y'_0) and the focus closest to the origin is used.

Hyperbola

Parametrization of a hyperbola also makes use of (E.3), in which now A'' and C'' must be of opposite sign. Assuming $A'' > 0$ and $C'' < 0$, define $a = \sqrt{1/A''}$, $b = \sqrt{-1/C''}$, $c = \sqrt{a^2 + b^2}$. Then the eccentricity is $e = c/a$, $l = b^2/a$, and $\theta_0 = 0$. The foci are located at $X'_f = X'_0 \pm c$, $Y'_f = Y'_0$, where X'_0 and Y'_0 are defined as for the ellipse. Again, the focus closest to the origin is used. The case when $A'' < 0$ and $C'' > 0$ is handled similarly.

Custom case

If the coordinates of the focus, X'_f, Y'_f, are far from the origin, the polar parametrization in (E.2) may become numerically ill-conditioned for representing 1D structures close to the origin. This is because X' and/or Y' will be obtained as a difference of two large numbers (e.g. X'_f and $r(\theta)\cos(\theta)$). As 1D structures must always lie on or inside the reference triangle in X, Y space, they will be close to the origin in X', Y' space, and hence susceptible to this ill-conditioning problem. This situation is alleviated by introducing a new "custom" parametrization specifically for those cases when (X'_f, Y'_f) is far from origin, as determined by comparing the distance to a maximum distance, $d_{\mathrm{f,max}}$, with default value $d_{\mathrm{f,max}} = 1000$.

The custom parametrization is based on a general polar representation of a conic, described in detail in [54], given by

$$\frac{1}{r(\theta)} = T_0 \sin(\theta) + T_1 \cos(\theta) \pm \sqrt{T_2 \sin(2\theta) + T_3 \cos(2\theta) + T_4}, \tag{E.4}$$

where the T_i are

$$\begin{aligned} &T_0 = \frac{-E'}{2F'}, \quad T_1 = \frac{-D'}{2F'}, \quad T_2 = \frac{D'E'}{4F'^2}, \\ &T_3 = \frac{D'^2 - E'^2 + 4C'F' - 4A'F'}{8F'^2}, \quad T_4 = \frac{D'^2 + E'^2 - 4C'F' - 4A'F'}{8F'^2}. \end{aligned} \tag{E.5}$$

The key in making this parametrization successful is choosing a well-conditioned origin for the polar representation, ideally close to $(X', Y') = (0, 0)$ but not too close to the conic. For example, if the conic intersects the chosen origin, the resulting polar parametrization would be singular due to $F' = 0$ in (E.5). In this work, the origin is chosen as the point furthest away from the conic out of a set that includes all vertices of the reference triangle vertices as well as its centroid.

Appendix F

Geometry Interpolation Properties of Quadratic Patches

This appendix compares two geometry representation techniques in both two and three dimensions: linear panels/patches and quadratic panels/patches. The question of interest is how accurately these techniques represent a curved geometry. In two dimensions, a linear panel representation is compared to a quadratic panel representation for a circle. In three dimensions, a linear patch representation is compared to a quadratic patch representation for a sphere.

Two dimensions

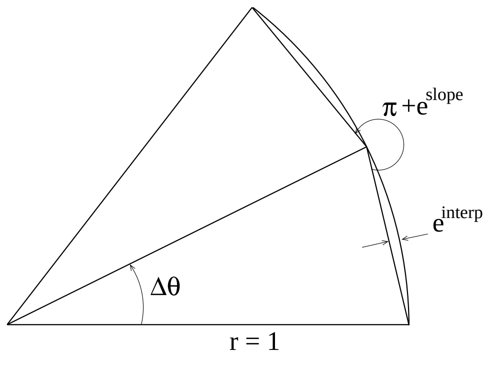
A section of a linear paneling of a circle of radius 1 is shown in Figure F-1a. The panels are placed at angle intervals of $\Delta\theta$. Two error values between the geometry and the paneling are of interest: the interpolation error, e^{interp}, which is the maximum distance between the panels and the true geometry; and the slope error, e^{slope}, which is the deviation of the angle between two panels from π. For a linear panel, assuming the panel endpoints are placed on the geometry, the maximum interpolation error occurs at angle $\Delta\theta/2$,

$$e^{\text{interp}} = 1 - \cos\Delta\theta/2 \approx \frac{\Delta\theta^2}{4}, \qquad \text{for small } \Delta\theta. \tag{F.1}$$

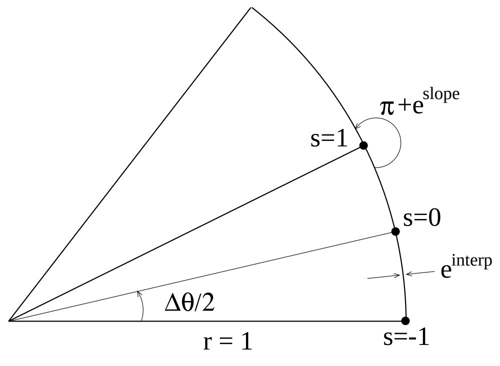
The slope error is simply the interval angle,

$$e^{\text{slope}} = \Delta\theta. \tag{F.2}$$

Compared to a linear panel, a quadratic panel contains an additional node. Figure F-1b



(a) Linear panels



(b) Quadratic panels

Figure F-1: Interpolation and slope error definitions for an arc of a circle in two dimensions. Nodes of the panels are assumed to be positioned on the true geometry.

depicts a standard quadratic panel with the additional node placed at $\Delta\theta/2$. The quadratic panel coordinates can be written as

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

where $\mathbf{x}_j$ are coordinates of the three nodes associated with the panel, $\phi_j(s)$ are quadratic Lagrange interpolant functions, and $s \in [-1, 1]$ is the interpolating parameter. The interpolation and slope errors for quadratic panels are calculated numerically in this example.

Consider a typical quadratic paneling using $\Delta\theta = \pi/12$, which corresponds to 12 panels (25 degrees of freedom, DOF) for a semi-circle. The result of a numerical calculation of e^{interp} along a panel is shown in Figure F-2. As expected, the interpolation error is zero at the three on-geometry locations ($s = -1, s = 0, s = 1$). The maximum interpolation is $e^{\text{interp}} = 9.1 \times 10^{-6}$. The required $\Delta\theta$ for a linear paneling to produce the same interpolation error is calculated from (F.1) to be $\Delta\theta = .006\text{rad}$, which corresponds to about 520 linear panels (521 DOF) for a semi-circle. Thus, to produce the same e^{interp}, a linear paneling requires over 40 times more patches (20 times more DOF).

For the same quadratic paneling with $\Delta\theta = \pi/12$, the slope error is calculated to be .0011rad. From (F.2), a linear paneling with the same slope error requires $\pi/.0011 = 2800$ panels. This is a factor of 230 increase in the number of panels, and a factor of 110 increase in the DOF.

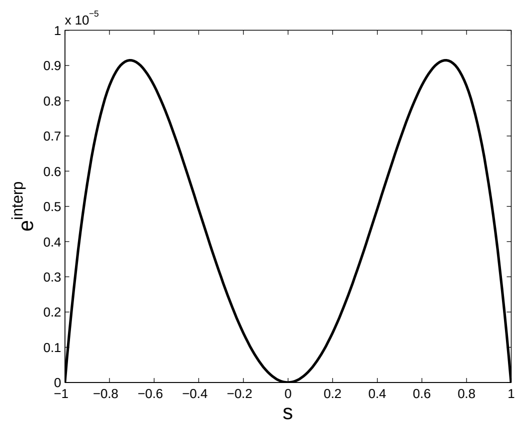


Figure F-2: Geometry interpolation error along a quadratic panel for $\Delta\theta = \pi/12$. The maximum interpolation error is 9.1×10^{-6}.

Three dimensions

In three dimensions, linear triangular patches are compared to quadratic triangular patches, introduced in Chapter 5, for representing a sphere. For analysis, one equilateral triangle patch is considered, as illustrated in Figure F-3. The side length of the triangle is d, and the radius of the sphere is set to 1. Thus, the patch count for a hemi-sphere is approximately

$$N_{\text{patch}} \approx \frac{2\pi}{A_{\text{triangle}}} = \frac{2\pi}{d^2\sqrt{3}/4} = \frac{8\pi}{d^2\sqrt{3}}. \tag{F.4}$$

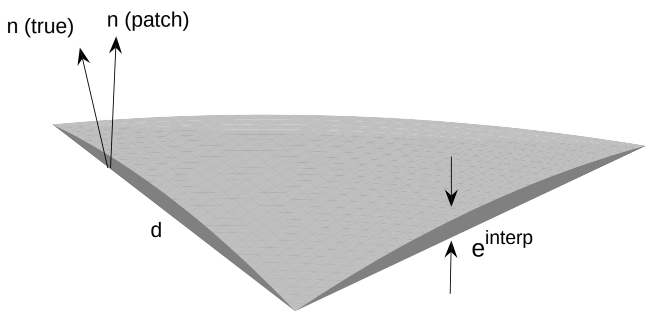


Figure F-3: Measurement of interpolation error and slope error for a linear patch (dark gray) on a sphere surface (light gray).

For a linear patch, placing the vertices on the geometry results in a maximum interpolation error at the centroid of the equilateral triangle:

$$e^{\text{interp}} = 1 - \sqrt{1 - \frac{d^2}{3}}. \qquad \text{(F.5)}$$

The slope error is measured as twice the maximum angle between the patch normal and the true normal along an edge of the patch. For a linear patch, in which the patch normal $\mathbf{n}_{\text{patch}}$ is constant (and exact at the centroid), a straightforward calculation yields that the maximum slope error occurs at the patch vertices, where

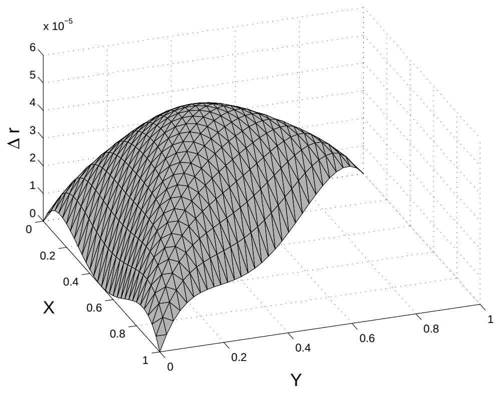
$$e^{\text{slope}} = \frac{2d}{\sqrt{3}} \text{ rad}. \qquad \text{(F.6)}$$

Quadratic patches require the placement of three additional nodes at the midpoints of the triangle edges. In this analysis, these high-order nodes are obtained by projecting the edge midpoints radially outward to the surface of the sphere. The resulting patch coordinates are given by (5.1), which is used to numerically evaluate the maximum interpolation and slope errors.

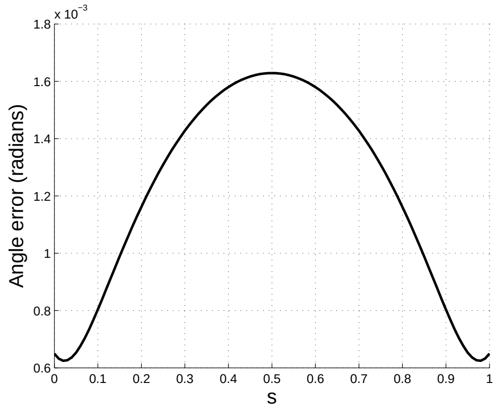
Consider a quadratic patch representation of a hemi-sphere using $d_{\text{quad}} = \pi/12$, which corresponds to about 12 patches along each half great-circle. The total number of patches is found by (F.4) to be $N_{\text{patch}} = 210$. The interpolation error over one quadratic patch is plotted in Figure F-4a. As expected, the patch vertices and edge midpoints are pinned to zero interpolation error, and the maximum deviation occurs at the centroid. In this case $e^{\text{interp}} = 5.0 \times 10^{-5}$. Using (F.5), the linear patch length, d_{lin}, required for the same interpolation error is found to be $d_{\text{lin}} = .0173$, so that the linear-to-quadratic patch number ratio is given by $d_{\text{quad}}^2/d_{\text{lin}}^2 = 230$. Thus, for the hemisphere, 48000 linear patches are required compared to 210 quadratic patches for the same geometry interpolation error.

A DOF comparison is obtained by noting that the number of vertices is approximately half the number of patches, while the number of edges is approximately 1.5 times the number of patches. Thus, a quadratic patch representation, with DOF not only on vertices but also on edges, requires approximately $2N_{\text{patch}}$ DOF, whereas a linear patch representation requires approximately $N_{\text{patch}}/2$ DOF. 210 quadratic patches therefore translates to 420 DOF, while 48000 linear patches translates to 24000 DOF – still a very substantial increase.

As in two dimensions, the slope error becomes even more restrictive. A plot of the slope error along an edge of a quadratic patch is given in Figure F-4b. The slope error is greatest at the edge midpoint, where $e^{\text{slope}} = .00326$rad in this case. Using (F.6), the linear patch length required for the same slope error is $d_{\text{lin}} = .00282$, which corresponds to a patch number ratio of $d_{\text{quad}}^2/d_{\text{lin}}^2 = 8600$. Therefore, for the hemisphere, a linear



(a) Interpolation error over patch



(b) Slope error along edge

Figure F-4: Interpolation and slope errors for a quadratic patch. The interpolation error (a) is shown over the entire patch in patch reference space. The slope error (b) is shown along one edge.

patch representation requires over 1.8 million patches to achieve the same slope error (0.9 million DOF). Such a high number of linear patches required to keep the slope error low suggests that a linear geometry representation may be overly expensive for $p > 1$ methods, which become increasingly sensitive to slope errors (i.e. geometry corners) as the order increases. Of course, as p and the slope accuracy requirement increase, even higher-order panels/patches will become more efficient. Nevertheless, for moderate solution interpolation orders, quadratic panels and patches will often be suitable and certainly more efficient than linear representations.

Bibliography

[1] M. J. Aftosmis. Solution adaptive Cartesian grid methods for aerodynamic flows with complex geometries. In *von Karman Institute for Fluid Dynamics, Lecture Series 1997-02*. Rhode-Saint-Genése, Belgium, Mar. 3-7, 1997.

[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] M. J. Aftosmis, M. J. Berger, and J. J. Alonso. Applications of a Cartesian mesh boundary-layer approach for complex configurations. AIAA Paper 2006-0652, 2006.

[4] T. Barth, and M. Larson. A posteriori error estimates for higher order Godunov finite volume methods on unstructured meshes. In R. Herban, and D. Kröner, editors, *Finite Volumes for Complex Applications III*, London, 2002. Hermes Penton.

[5] F. Bassi, and S. Rebay. High-order accurate discontinuous finite element solution of the 2-D Euler equations. *Journal of Computational Physics*, 138:251–285, 1997.

[6] F. Bassi, and S. Rebay. GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations. In K. Cockburn, and Shu, editors, *Discontinuous Galerkin Methods: Theory, Computation and Applications*, pages 197–208. Springer, Berlin, 2000.

[7] F. Bassi, and S. Rebay. Numerical evaluation of two discontinuous Galerkin methods for the compressible Navier-Stokes equations. *Int J Numer Meth Fluids*, 40:197–207, 2002.

[8] 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.

[9] 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.

[10] M. J. Berger, and R. J. Leveque. An adaptive Cartesian mesh algorithm for the Euler equations in arbitrary geometries. AIAA Paper 1989-1930, 1989.

[11] K. S. Bey, and J. T. Oden. *hp*-version discontinuous Galerkin methods for hyperbolic conservation laws. *Comput. Methods. Appl. Mech. Engrg.*, 133:259–286, 1996.

[12] H. Borouchaki, P. George, F. Hecht, P. Laug, and E. Saltel. Mailleur bidimensionnel de Delaunay gouverné par une carte de métriques. Partie I: Algorithmes. INRIA-Rocquencourt, France. Tech Report No. 2741, 1995.

[13] A. Brandt. *Guide to Multigrid Development.* Springer-Verlag, 1982.

[14] F. Brezzi, L. Marini, and E. Süli. Discontinuous Galerkin methods for first-order hyperbolic problems. *Math. Models Methods Appl. Sci.*, 14:1893–1903, 2004.

[15] O. Brodersen, and A. Stürmer. Drag prediction of engine-airframe interference effects using unstructured Navier-Stokes calculations. AIAA Paper 2001-2414, 2001.

[16] D. Calhoun, and R. J. LeVeque. A Cartesian grid finite-volume method for the advection-diffusion equation in irregular geometries. *Journal of Computational Physics*, 157:143–180, 2000.

[17] M. J. Castro-Diaz, F. Hecht, B. Mohammadi, and O. Pironneau. Anisotropic unstructured mesh adaptation for flow simulations. *International Journal for Numerical Methods in Fluids*, 25:475–491, 1997.

[18] D. K. Clarke, M. D. Salas, and H. A. Hassan. Euler calculations for multielement airfoils using Cartesian grids. *AIAA Journal*, 24(3):353, 1986.

[19] B. Cockburn, and C.-W. Shu. Runge-Kutta discontinuous Galerkin methods for convection-dominated problems. *Journal of Scientific Computing*, pages 173–261, 2001.

[20] W. J. Coirier, and K. G. Powell. Solution-adaptive cut-cell approach for viscous and inviscid flows. *AIAA Journal*, 34(5):938–945, 1996.

[21] 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.

[22] D. De Zeeuw, and K. G. Powell. An adaptively refined Cartesian mesh solver for the Euler equations. *Journal of Computational Physics*, 104:56–68, 1993.

[23] L. Diosady, and D. Darmofal. Discontinuous Galerkin solutions of the Navier-Stokes equations using linear multigrid preconditioning. AIAA Paper 2007-3942, 2007.

[24] K. J. Fidkowski. A high-order discontinuous Galerkin multigrid solver for aerodynamic applications. MS thesis, M.I.T., Department of Aeronautics and Astronautics, June 2004.

[25] K. J. Fidkowski, and D. L. Darmofal. Output-based adaptive meshing using triangular cut cells. M.I.T. Aerospace Computational Design Laboratory Report. ACDL TR-06-2, 2006.

[26] L. Formaggia, S. Micheletti, and S. Perotto. Anisotropic mesh adaptation with applications to CFD problems. In H. A. Mang, F. G. Rammerstorfer, and J. Eberhardsteiner, editors, *Fifth World Congress on Computational Mechanics*, Vienna, Austria, July 7-12 2002.

[27] H. Forrer, and R. Jeltsch. A higher-order boundary treatment for Cartesian-grid methods. *Journal of Computational Physics*, 140:259–277, 1998.

[28] N. T. Frink. Test case results from the 3rd AIAA drag prediction workshop. NASA Langley, 2007. `http://aaac.larc.nasa.gov/tsab/cfdlarc/aiaa-dpw/Workshop3/final_results_jm.tar.gz`.

[29] R. L. Gaffney, M. D. Salas, and H. A. Hassan. Euler calculations for wings using Cartesian grids. AIAA Paper 1987-0356, 1987.

[30] M. Giles, and N. Pierce. Adjoint error correction for integral outputs. In *Lecture Notes in Computational Science and Engineering: Error Estimation and Adaptive Discretization Methods in Computational Fluid Dynamics*, volume 25. Springer, Berlin, 2002.

[31] 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.

[32] W. G. Habashi, J. Dompierre, Y. Bourgault, D. Ait-Ali-Yahia, M. Fortin, and M.-G. Vallet. Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent CFD. part I: general principles. *Int. J. Numer. Meth. Fluids*, 32:725–744, 2000.

[33] R. Haimes. CAPRI: Computational analysis programming interface, a solid modeling based infra-structure for engineering analysis and design. CAPRI user's guide, MIT, Revision 1.00, 2000.

[34] K. Harriman, P. Houston, B. Senior, and E. Süli. hp-version discontinuous Galerkin methods with interior penalty for partial differential equations with nonnegative characteristic form. Technical Report Technical Report NA 02/21, Oxford University Computing Lab Numerical Analysis Group, 2002.

[35] R. Hartmann, and P. Houston. Goal-oriented a posteriori error estimation for multiple target functionals. In T. Hou, and E. Tadmor, editors, *Hyperbolic Problems: Theory, Numerics, Applications*, pages 579–588. Springer-Verlag, 2003.

[36] R. Hartmann, and P. Houston. Adaptive discontinuous Galerkin finite element methods for the compressible Euler equations. *Journal of Computational Physics*, 183(2):508–532, 2002.

[37] K. J. Hill. Matrix-based ellipse geometry. In A. W. Paeth, editor, *Graphics Gems V*, pages 72–77. Academic Press, San Diego, CA, 1995.

[38] P. Houston, R. Hartmann, and E. Süli. Adaptive discontinuous Galerkin finite element methods for compressible fluid flows. In M. Baines, editor, *Numerical Methods for Fluid Dynamics VII*, volume 8, page 341, 2001.

[39] P. Houston, and E. Süli. Error estimation and adaptive discretization methods in computational fluid dynamics. In T. Barth, and H. Deconinck, editors, *Error Estimation and Adaptive Discretization Methods in Computational Fluid Dynamics*, pages 269–344. Springer-Verlag, Heidelberg, Lecture Notes in Computational Science and Engineering Vol 25, 2002.

[40] P. Houston, and E. Süli. A note on the design of hp-adaptive finite element methods for elliptic partial differential equations. *Comput. Methods Appl. Mech. Engrg*, 194:229–243, 2005.

[41] S. L. Karman. SPLITFLOW: A 3d unstructured Cartesian/prismatic grid CFD code for complex geometries. AIAA Paper 1995-0343, 1995.

[42] K. R. Laflin, J. C. Vassberg, R. A. Wahls, J. H. Morrison, O. Brodersen, M. Rakowitz, E. N. Tinoco, and J.-L. Godard. Summary of data from the second AIAA CFD drag prediction workshop. AIAA Paper 2004-0555, 2004.

[43] P. R. Lahur, and Y. Nakamura. A new method for thin body problem in Cartesian grid generation. AIAA Paper 99-0919, 1999.

[44] P. R. Lahur, and Y. Nakamura. Anisotropic Cartesian grid adaptation. AIAA Paper 2000-2243, 2000.

[45] R. J. LeVeque. A large time step generalization of godunov's method for systems of conservation laws. *SIAM J.Numer. Anal.*, 22(6):1051–1073, 1985.

[46] R. J. LeVeque. High resolution finite volume methods on arbitrary grids via wave propagation. *Journal of Computational Physics*, 78:36–63, 1988.

[47] D. W. Levy, T. Zickuhr, J. Vassberg, S. Agrawal, R. A. Wahls, S. Pirzadeh, and M. J. Hemsch. Data summary from the First AIAA Computational Fluid Dynamics Drag Prediction Workshop. *Journal of Aircraft*, 40(5):875–882, 2003.

[48] J. 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.

[49] L. Machiels, J. Peraire, and A. Patera. A posteriori finite-element output bounds for the incompressible Navier-Stokes equations: Application to a natural convection problem. *Journal of Computational Physics*, 172:401–425, 2001.

[50] D. J. Mavriplis. An assessment of linear versus nonlinear multigrid methods for unstructured mesh solvers. *Journal of Computational Physics*, 175:302–325, 2001.

[51] D. J. Mavriplis. Results from the 3rd drag prediction workshop using the NSU3D unstructured mesh solver. AIAA Paper 2007-256, 2007.

[52] J. E. Melton, F. Y. Enomoto, and M. J. Berger. 3D automatic Cartesian grid generation for Euler flow. AIAA Paper 1993-3386-CP, 1993.

[53] J. H. Morrison, and M. J. Hemsch. Statistical analysis of CFD solutions from the third AIAA drag prediction workshop. AIAA Paper 2007-254, 2007.

[54] F. R. Moulton. *An Introduction to Celestial Mechanics. 2nd rev. ed.* Dover, New York, 1970.

[55] S. M. Murman, M. J. Aftosmis, and S. E. Rogers. Characterization of space shuttle ascent debris aerodynamics using CFD methods. AIAA Paper 2005-1223, 2005.

[56] M. Nemec, M. J. Aftosmis, and T. H. Pulliam. CAD-based aerodynamic design of complex configurations using a Cartesian method. Technical Report NAS-04-001, NASA, 2004.

[57] M. Nemec, M. Aftosmis, S. Murman, and T. Pulliam. Adjoint formulation for an embedded-boundary Cartesian method. AIAA Paper 2005-0877, 2005.

[58] T. A. Oliver. Multigrid solution for high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. MS thesis, M.I.T., Department of Aeronautics and Astronautics, August 2004.

[59] R. Pember, J. B. Bell, P. Colella, W. Y. Crutchfield, and M. L. Welcome. An adaptive Cartesian grid method for unsteady compressible flow in irregular regions. *Journal of Computational Physics*, 120:278–304, 1995.

[60] J. Peraire, M. Vahdati, K. Morgan, and O. C. Zienkiewicz. Adaptive remeshing for compressible flow computations. *Journal of Computational Physics*, 72:449–466, 1987.

[61] N. A. Pierce, and M. B. Giles. Adjoint recovery of superconvergent functionals from PDE approximations. *SIAM Review*, 42(2):247–264, 2000.

[62] J. W. Purvis, and J. E. Burkhalter. Prediction of critical Mach number for store configurations. *AIAA Journal*, 17(11):1170–1177, 1979.

[63] J. Quirk. An alternative to unstructured grids for computing gas dynamic flows around arbitrarily complex two dimensional bodies. Technical Report 92-7, ICASE, 1992.

[64] R. Rannacher. Adaptive Galerkin finite element methods for partial differential equations. *Journal of Computational and Applied Mathematics*, 128:205–233, 2001.

[65] P. L. Roe. Approximate Riemann solvers, parametric vectors, and difference schemes. *Journal of Computational Physics*, 43:357–372, 1981.

[66] P. E. Rubbert, J. E. Bussoletti, F. T. Johnson, K. W. Sidwell, W. S. Rowe, S. S. Samant, G. SenGupta, W. H. Weatherill, R. H. Burkhart, B. L. Everson, D. P. Young, and A. C. Woo. A new approach to the solution of boundary value problems involving complex configurations. In A. K. Noor, editor, *Computational Mechanics – Advances and Trends*, pages 49–84, 1986.

[67] H. Si. Tetgen: A quality tetrahedral mesh generator and three-dimensional Delaunay triangulator. Weierstrass Institute for Applied Analysis and Stochastics, 2005. `http://tetgen.berlios.de`.

[68] P. Solín, and L. Demkowicz. Goal-oriented hp-adaptivity for elliptic problems. *Comput. Methods Appl. Mech. Engrg.*, 193:449–468, 2004.

[69] G. Strang, and G. J. Fix. *An Analysis of the Finite Element Method.* Wellesley-Cambridge Press, 1988.

[70] B. A. Szabo. Estimation and control of error based on p convergence. In I. Babuska, O. C. Zienkiewicz, J. Gago, and E. R. de Oliveira, editors, *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, pages 61–78. John wiley & Sons Ltd., 1986.

[71] J. C. Vassberg, M. A. DeHaan, and T. J. Sclafani. Grid generation requirements for accurate drag predictions based on OVERFLOW calculations. AIAA Paper 2003-4124, 2003.

[72] D. A. Venditti. *Grid Adaptation for Functional Outputs of Compressible Flow Simulations*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2002.

[73] D. A. Venditti, and D. L. Darmofal. Grid adaptation for functional outputs: application to two-dimensional inviscid flows. *Journal of Computational Physics*, 176(1):40–39, 2002.

[74] D. A. Venditti, and D. L. Darmofal. Anisotropic grid adaptation for functional outputs: application to two-dimensional viscous flows. *Journal of Computational Physics*, 187(1):22–46, 2003.

[75] V. Venkatakrishnan, S. R. Allmaras, D. S. Kamenetskii, and F. T. Johnson. Higher order schemes for the compressible Navier-Stokes equations. AIAA Paper 2003-3987, 2003.

[76] B. Wedan, and J. C. South. A method for solving the transonic full-potential equation for general configurations. AIAA Paper 1983-1889, 1983.

[77] F. M. White. *Viscous Fluid Flow*. McGraw-Hill, Inc., 1974.

[78] D. P. Young, R. G. Melvin, M. B. Bieterman, F. T. Johnson, S. S. Samant, and J. E. Bussoletti. A higher-order boundary treatment for Cartesian-grid methods. *Journal of Computational Physics*, 92:1–66, 1991.

[79] X. D. Zhang, M.-G. Vallet, J. Dompierre, P. Labbe, D. Pelletier, J.-Y. Trepanier, R. Camarero, J. V. Lassaline, L. M. Manzano, and D. W. Zingg. Mesh adaptation using different error indicators for the Euler equations. AIAA Paper 2001-2549, 2001.

[80] O. C. Zienkiewicz, and J. Z. Zhu. Adaptivity and mesh generation. *International Journal for Numerical Methods in Engineering*, 32:783–810, 1991.