Boundary Treatment for Anisotropic All-Quad Mesh Adaptation through an L_p-CVT Approach

Chun Kit Calvin Li

Department of Mechanical Engineering
McGill University, Montreal

December 2023

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Master of Science

Abstract

In computational fluid dynamics (CFD), achieving highly accurate and efficient simulations often demands the application of adaptive mesh refinement strategies. In particular, all-quad meshes are widely acclaimed for their ability to reduce the complexity of tensor-product formulations compared to triangles, making them a preferred choice for the Discontinuous Galerkin Method. Nevertheless, the generation and adaptation of such meshes near intricate boundaries continue to present formidable challenges, especially when confronted with anisotropic flow characteristics. Anisotropy in flow occurs when specific flow directions exhibit distinct characteristics or gradients compared to others such as in the vicinity of shockwaves. Accurate representation of such anisotropic features is crucial in comprehending complex fluid flow behaviors. This work presents an approach to address these challenges through boundary treatment for all-quad mesh adaptation using the L_p-CVT (L_p-Centroidal Voronoi Tessellation) framework. While our primary focus lies in applying this approach to simulate a transonic flow over a NACA0012 airfoil within a CMesh, the secondary focus is on testing the boundary treatment method on analytical test cases to ascertain its versatility. Additionally, a comparative study against a conventional test case without boundary treatment is conducted to emphasize the importance of boundary treatment in capturing anisotropic features. Lastly, the compatibility of this method with the Discontinuous Galerkin Method highlights its applicability in the context of high-order numerical discretization for compressible flows, ultimately advancing our ability to model and understand complex fluid flow phenomena.

Résumé

En dynamique des fluides numérique (CFD), la réalisation de simulations très précises et efficaces nécessite souvent l'application de stratégies adaptatives de raffinement du maillage. En particulier, les maillages tout quad sont largement reconnus pour leur capacité à réduire la complexité des formulations de produits tensoriels par rapport aux triangles, ce qui en fait un choix privilégié pour la méthode Galerkin discontinue. Néanmoins, la génération et l'adaptation de tels maillages à proximité de frontières complexes continuent de présenter de formidables défis, en particulier lorsqu'elles sont confrontées à des caractéristiques d'écoulement anisotropes. L'anisotropie de l'écoulement se produit lorsque des directions d'écoulement spécifiques présentent des caractéristiques ou des gradients distincts par rapport à d'autres, par exemple à proximité d'ondes de choc. Une représentation précise de ces caractéristiques anisotropes est cruciale pour comprendre les comportements complexes d'écoulement des fluides. Ce travail présente une approche pour relever ces défis grâce au traitement des limites pour l'adaptation du maillage tout quad à l'aide du cadre L_p-CVT (L_p-Centroidal Voronoi Tessellation). Alors que notre objectif principal consiste à appliquer cette approche pour simuler un écoulement transsonique sur un profil aérodynamique NACA0012 au sein d'un CMesh, l'objectif secondaire est de tester la méthode de traitement des limites sur des cas de tests analytiques afin de vérifier sa polyvalence. De plus, une étude comparative avec un cas de test conventionnel sans traitement des limites est menée pour souligner l'importance du traitement des limites dans la capture des caractéristiques anisotropes. Enfin, la compatibilité de cette méthode avec la méthode Discontinue Galerkin met en évidence son applicabilité dans le contexte de la discrétisation numérique d'ordre élevé pour les écoulements compressibles, améliorant ainsi notre capacité à modéliser et à comprendre les phénomènes complexes d'écoulement des fluides.

Acknowledgements

I would like to first thank Professor Siva Nadarajah for his guidance, support, and advice throughout completing this research thesis. He has shown me on numerous occasions how to critically think and ask questions regarding the smallest details to further answer some of the ongoing loopholes in the field of Mesh Adaptation. His dedication to detail has driven me to always pursue excellence in my work, which is key to the success of the research results shown in this thesis.

Second, I would like to thank my parents, my two brothers, and my girlfriend for their constant support in allowing me to always have priorities in my academic endeavors.

Third, I would like to thank all my peers from the Computational Aerodynamics Lab. It has been a fruitful 2 years with the group, and I am grateful to have had their companionship throughout the completion of this research project. Notably, I am thankful to Pranshul Thakur, Alexander Cicchino, and Julien Brillon for their contributions to the success of this project.

Lastly, I want to thank Professor Siva Nadarajah and the Department of Mechanical Engineering at McGill University for their financial support throughout the completion of this degree.

Table of Contents

Abstract i

Résumé ii

Acknowledgements iv

List of Figures x

List of Tables xi

Nomenclature **xii**

1 Introduction **1**

1.1 Overview of Mesh Adaptation Techniques 4

1.1.1 *A Priori* and *A Posteriori* Error Estimators 4

1.1.2 Riemannian Metric and Frame Fields 6

1.1.3 Metric-based Mesh Adaptation 7

1.2 Thesis Overview 9

2 Mesh Generation **10**

2.1 Structured vs Unstructured Mesh 10

2.2 Computational vs Physical Domain 15

2.2.1 Transformation Matrix 16

2.3 Two-Dimensional Triangular Mesh Generator 19

2.3.1 Delaunay Triangulation 19

2.3.2 Voronoi Diagram 21

2.4 Two-Dimensional Algebraic and PDEs based Quadrilateral Structured Mesh Generator 23

2.4.1 Transfinite Interpolation 23

2.4.2 Hermite Interpolation 26

2.4.3 Parabolic Equation Approach 28

2.4.4 Elliptic Equation Approach 29

2.4.5 Hyperbolic Equation Approach 33

2.4.6 Solution to the Hyperbolic Equation Approach 34

2.4.7 Artificial Dissipation 37

2.5 Metric based Inner Product, Norm, and Distance Definition 40

2.5.1 Euclidean Metric Space 41

2.5.2 Riemmanian Metric Space 41

2.6 Frame Fields 44

2.7 Metric-based Mesh Adaptation 47

2.8 L_p-CVT Mesh Generator 49

2.8.1 Local Facet Energy Computation 49

2.8.2 Total Energy and Derivative Computation 50

2.8.3 Quasi-Unit Mesh 55

2.9 Simplified Boundary Reconstruction of L_p-CVT 56

2.10 Boundary Nodes Distribution 58

2.11 Moving Boundary Formulation for L_p-CVT 59

2.11.1 Modified Total Energy and Derivative Computation 59

2.12 All-Quad Mesh Generator 61

2.13 Overview of L_p-CVT algorithm 61

3 Numerical Methods **63**

3.1 Discontinuous Galerkin Method 64

3.2 Discrete Error Minimization 66

3.3 Adjoint Method 67

3.3.1 Dual-Weight Residual 69

4 Continuous Mesh Model and Error Estimates 72

4.1 Continuous Mesh Model 72

4.1.1 Continuous Element Definition of Area and Density 73

4.1.2 Continuous Mesh Characterization 73

4.2 Continuous Error Model 74

4.2.1 Local Interpolation Error 74

4.2.2 Global Interpolation Error 77

4.3 Goal-Oriented Approach 78

5 Results 81

5.1 Analytical Metric Field 81

5.1.1 Constant Metric Field 82

5.1.2 Quadratic Metric Field 88

5.2 Discrete Frame Field 92

5.2.1 S-Shock Adaptation 92

5.2.2 Boundary Layer Adaptation 101

5.3 Euler Test Case 107

5.3.1 Transonic NACA0012 Steady State Test Case 108

6 Conclusion 114

6.1 Future Work 115

List of Figures

1.1 Two-Dimensional CH10 Airfoil Structured Mesh 2

1.2 Mesh Adaptation Techniques - Coupled 4

1.3 Mesh Adaptation Techniques - Decoupled 4

2.1 Two-Dimensional Structured Grid with 2^{nd} order polynomial growth function along the y-axis. 12

2.2 Two-Dimensional Delaunay Triangulation over a NACA0012 airfoil bounded by a circular domain produced by DistMesh. 13

2.3 Leading Edge of NACA0012 airfoil bounded by a circular domain produced by DistMesh. 14

2.4 Left: Physical Domain — Right: Computational Domain. 16

2.5 Structured CMesh Grid with quadrilateral cells generated using a Hyperbolic Grid Generator 18

2.6 CDT generated by DistMesh inside a bounded domain 20

2.7 Geometric duality between Delaunay Triangulation and Voronoi Diagram . 21

2.8 Comparison of regular Voronoi cell and Centroidal Voronoi cell 22

2.9 Transfinite Interpolation of a rectangle with a semicircle. 25

2.10 Transfinite Interpolation in three dimensional space. 25

2.11 Solution to Heat Equation - Isothermal lines. 29

2.12 Superimposed solution of Heat Equation. 30

2.13 Clustering effect by introducing $Q(\xi, \eta)$. 32

2.14 Example of Hyperbolic Equation Grid Generation at the leading edge of airfoil 38
2.15 Transfinite Interpolation 39
2.16 Hermite Interpolation 39
2.17 Laplace Equation Solution 39
2.18 Poisson Equation Solution 39
2.19 Unit ellipse at each position depicted by red points 44
2.20 Frame field Mapping 45
2.21 Comparison of various L$_p$-norm on a Circle 47
2.22 Metric tensor ellipse with a single corresponding generated triangle 48
2.23 Metric tensor ellipse with possible configuration of generated triangles . . . 48
2.24 Voronoi Cell with 6 facets - Left: Full Volume Voronoi Cell, Right: Clipped Boundary Voronoi Cell 49
2.25 Triangle with 3 Bisectors 52
2.26 Example of Voronoi Diagram with clipped boundary 54
2.27 Example of Edge Splitting 55
2.28 Example of Edge Merging 56
2.29 Reconstruction of Boundary Voronoi Cell 57
2.30 Moving boundary formulation for L_p-CVT 59
2.31 Flowchart showing the L_p-CVT algorithm 62

3.1 2^{nd} order triangle and quadrilateral elements 65

4.1 H^1 patchwise reconstruction 76
4.2 Flowchart showing the flow solver coupled with the L_p-CVT algorithm . . . 80

5.1 Comparison of clipped and reconstructed boundary Voronoi diagram based on a constant metric field 82
5.2 Reconstructed boundary with ghost cells 83
5.3 Constant Metric Field Result Summary 85

5.4 Constant Metric Field All-Quad Mesh Summary 86
5.5 Constant Metric Field Result with Irregular Boundary 87
5.6 Quadratic Metric Field Result Summary 89
5.7 All-Quad Output Mesh and Energy Gradient 90
5.8 Feature-based Initial S-Shock Solution 92
5.9 Summary of S-Shock Case Results Set 1 - Moving Boundary 97
5.10 Summary of S-Shock Case Results set 2 - Moving Boundary 98
5.11 Summary of S-Shock Case Results Set 1 - Static Boundary 99
5.12 Summary of S-Shock Case Results Set 2 - Static Boundary 100
5.13 Summary of S-Shock Case Results 101
5.14 Summary of Boundary Layer Case Results Set 1 - Moving Boundary 103
5.15 Summary of Boundary Layer Case Results set 2 - Moving Boundary 104
5.16 Summary of Boundary Layer Case Results Set 1 - Static Boundary 105
5.17 Summary of Boundary Layer Case Results Set 2 - Static Boundary 106
5.18 Summary of Boundary Layer Case Results 107
5.19 Moving Boundary L_p-CVT Mesh Adaptation Cycles 111
5.20 Static Boundary L_p-CVT Mesh Adaptation Cycles 112
5.21 Final Output Mesh Result Summary 113

List of Tables

5.1 Energy gradient value for clipped boundary ($\frac{dE}{dx_i}_C$) and reconstructed boundary ($\frac{dE}{dx_i}_R$) at the bottom and right side boundary 83

5.2 Summary of convergence for Quadratic Metric All-Quad Mesh and $\|\nabla E\|_2$. 91

5.3 Summary of reconstructed length for 25 boundary points based on the nearest frame field of the initial S-Shock solution 94

Nomenclature

Physical and computational domain

x	Physical domain horizontal location
y	Physical domain vertical location
ξ	Computational domain horizontal location
η	Computational domain vertical location
$\boldsymbol{J}_1$	Computational domain Jacobian matrix
$\boldsymbol{J}_2$	Physical domain Jacobian matrix
J	Jacobian determinant

Voronoi and Delaunay diagram

$\mathcal{T}_i$	ith Delaunay triangle
$\mathcal{V}_i$	ith Voronoi cell
P_i	ith Voronoi site
g_i	Centroid of ith Voronoi cell

Algebraic and PDEs based quadrilateral structured mesh generation

i	Horizontal direction traversal index

j	Vertical direction traversal index
$P(\xi, \eta)$	Horizontal forcing function
$Q(\xi, \eta)$	Vertical forcing function
$F_{i,j}$	Cell area function
s_1	Spacing between first and second boundary grid line
ϵ	Stretching ratio
β	Artificial dissipation factor

Riemannian metric space

$\mathcal{M}$	Metric tensor
$l_{\mathcal{M}}$	Distance under metric tensor
λ_1, λ_2	Eigenvalues of metric tensor M

Frame field

f_p	Frame field at point p
V	Frame field linear mapping
e_1, e_2	Principal axis of frame field
θ_1, θ_2	Orientation of frame field
h_1, h_2	Length of principal axis of frame field
h	Average size of frame field
ρ	Anisotropy
$R(\theta)$	Rotation matrix

L_p-CVT mesh generator

$\mathcal{T}_h$	Mesh triangulation
I_k	Local energy
f_i	ith facet
$\boldsymbol{T(\xi)}$	Reference triangle
E_{L_p}	L_p-CVT total energy
$M(y)$	Background metric field
$\boldsymbol{e_i}$	Mesh edge
l_{min}	Minimum edge length for merging
l_{min}	Maximum edge length for splitting
$l_{b,i}$	Local reconstructed boundary length
l_r	Simplified reconstructed boundary length
L_M	Edge length under metric space

Discontinuous Galerkin method

$\boldsymbol{u}$	Flow solution
$\boldsymbol{R}(u)$	Residual
$\boldsymbol{F}(u)$	Flux vector
$\hat{\boldsymbol{F}}(u)$	Numerical flux vector
$\boldsymbol{S}(u)$	Source term
u_h	Discrete flow solution

ϕ_h	Shape function
ψ_h	Test function
p_k	Local polynomial order
$N_k(p_k)$	Total number of DOFs
Ω	Physical domain
Ω_k	Local domain
Ω_h	Computational domain

Discrete error minimization

E_h	Discrete error
Π_{hp}	Optimal projection operator
C	Target complexity

Adjoint method

$\mathcal{J}(u)$	Functional of interest
ψ	Adjoint variable
ψ_h	Discrete adjoint variable
$\mathcal{L}$	Lagrange multiplier
I_h^H	Projection operator from coarse to fine grid
η_k	Local dual-weighted residual

Continuous mesh model

$A(\boldsymbol{x})$	Local element area

$d(\boldsymbol{x})$	Local element density
$N(V)$	Total number of cells
$\mathcal{N}(V, \mathcal{P})$	Total number of degrees of freedom
$\mathcal{P}(\boldsymbol{x})$	Polynomial distribution

Continuous error model

$e^{int}_{\tilde{\boldsymbol{x}},p}(\boldsymbol{x})$	Local interpolation error of degree p at location x
A_1	Maximum of p+1 directional derivative in direction 1
A_2	Maximum of p+1 directional derivative in direction 2
$\boldsymbol{\xi}_1$	Unit vector of direction 1
$\boldsymbol{\xi}_2$	Unit vector of direction 2
u_h^+	Enriched solution through H^1 patchwise reconstruction
e	Local continuous error estimator
$\mathcal{E}$	Global continuous error estimator

Goal-oriented approach

I_k	Target cell area
I_k^c	Current cell area
α_k	Scaling factor
ξ_k	Logarithmic scaling factor
r_{max}	Maximum refinement factor
c_{max}	Maximum coarsening factor

Chapter 1

Introduction

In the early phases of a novel aerospace conceptual design, significant attention is dedicated to assembling all of the available computational analysis tools that can be leveraged. Through the results generated by these tools, engineers can better understand the capability and limitations of their designs during various simulated flight conditions. In particular, non-dimensional aerodynamic forces such as coefficient of lift and drag are of interest because these parameters allow for the comparison between various designs (such as different wing, or engine configurations). This eventually leads to the optimization of the design allowing the engineer to specifically select the best-suited design for the target operation envelope. Amongst the available tools, Computational Fluid Dynamics (CFD) excels in this domain by efficiently calculating and forecasting the flow's behavior across a domain, empowering engineers to leverage these outcomes for optimizing their designs to meet engineering expectations. In practice, two main building blocks are required for building a CFD simulation, and they are namely a mesh and a flow solver. However, some numerical methods can be independent of the mesh which are known as mesh-free methods, where the solution point, i.e. a fluid particle, can move freely in the spatial domain through interactions with its neighbors which is known as the Lagrangian approach [1]. A mesh is considered a prerequisite for the flow solver, and hence, must be generated beforehand. Mesh generation consists of discretizing a domain prescribed by a geometry which often acts as the boundary over which the flow will be simulated, i.e., an airfoil, as shown in Figure 1.1. Once the mesh is obtained, the flow solver can utilize the mesh to perform calculations over each cell to predict and converge the solution with a certain level of acceptance of the error.

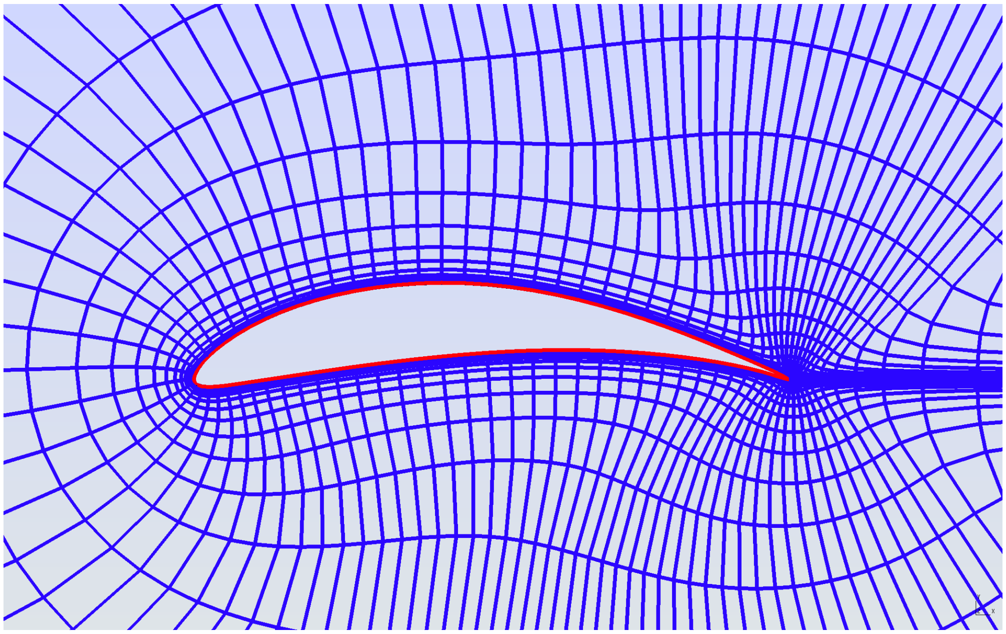


Figure 1.1: Two-Dimensional CH10 Airfoil Structured Mesh

Despite recent advancements in numerical models, there are still limitations on the performance of these solvers due to the mesh utilized for computations. This has been identified as one of the open topics that require improvement in NASA's CFD Vision 2030 Study report [2] which is set to be resolved in the upcoming years. Another source of error in the output solution arises from significant changes in flow features. Typically, these are known as anisotropic flow features characterized by directional notions that must be resolved for an accurate solution to occur. For instance, when the flow encounters a shockwave, there is an abrupt deceleration resulting in a substantial change in the velocity gradient. This creates a challenging region for the solver to converge, as accurately resolving the velocity change of the flow passing through the shockwave becomes increasingly difficult. Therefore, solution or mesh points must be strategically placed to identify areas where additional points are needed to accurately depict the solution. One way to achieve this is to employ cells of different sizes and orientations in those regions which can help the solver converge much faster compared to a uniformly distributed mesh. This is because anisotropic cells aid the solver in resolving anisotropic flow features more effectively by aligning the cells with the anisotropic flow features of the solution. Additionally, accurate representation of the shockwave is very difficult from a static boundary mesh configuration (i.e. boundary points do not move), hence boundary treatments are key to allowing the mesh to have flexibility at the boundary to thereby better depict such flow features. A key improvement to resolving the solution in those regions is to make use of mesh adaptation techniques [3, 4, 5, 6] which adapts the mesh based on the need to better resolve the accuracy of the solution or feature of interest at those locations.

1.1 Overview of Mesh Adaptation Techniques

Over the past two decades, mesh adaptation techniques have gained significant research interest within the CFD community. This is due to its ability to reduce the computational cost of a flow solver while maintaining or increasing the level of accuracy in the solution. In simple words, mesh adaption seeks to generate an optimal mesh to control the error of a solution [7]. Mesh adaptation can, in general, be separated into two different classes when coupled with a flow solver, as depicted in the following Figure 1.3.

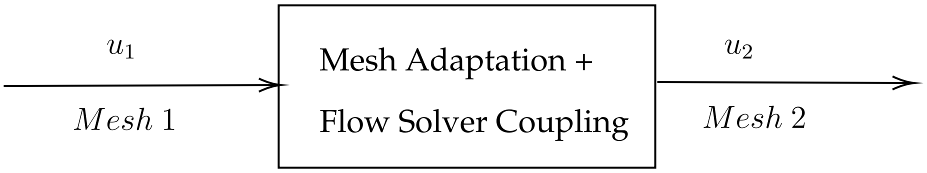


Figure 1.2: Mesh Adaptation Techniques - Coupled

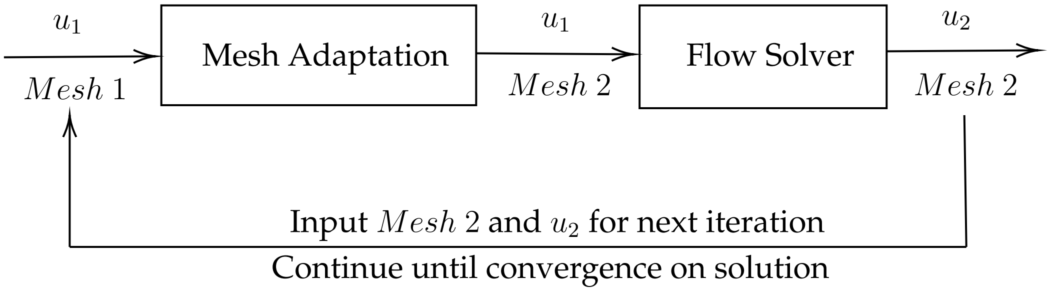

u_1
$Mesh\ 1$
Mesh Adaptation
u_1
$Mesh\ 2$
Flow Solver
u_2
$Mesh\ 2$
Input $Mesh\ 2$ and u_2 for next iteration
Continue until convergence on solution

Figure 1.3: Mesh Adaptation Techniques - Decoupled

1.1.1 *A Priori* and *A Posteriori* Error Estimators

The error indicator is a map where each cell contains information from the error of the solution. The goal of the error indicator is to quantify the error that is produced in the solution and allow the mesh adaptation algorithm to identify regions of high and low error and adapt its mesh accordingly. This information is computed on the original grid, or in some cases, on a projected fine mesh through patch reconstruction schemes, and then, provided to the mesh adaptation generator to produce a new mesh. As mentioned previously, two error estimators are possible, and they are either *a priori* or *a posteriori* error estimators. The term *a priori* stands for ″prior″ or ″beforehand″ and the term *a posteriori* stands for ″post″ or ″after″.

Just as the name implies, *a priori* error estimator can be computed before the solution converges through manipulations of the governing equation, hence it does not require the solution of the flow solver and relies only on mathematical expressions to predict the region of interest for the mesh adaptation to occur. The benefit of such a method is that the error estimator can be determined even before solving for the solution, however, the expression itself requires derivations which are a function of the mesh as well as the solution, u, and can often be cumbersome to resolve. Also, this presents a difficulty in numerical simulations where the exact solution is rarely available, and therefore, *a priori* error estimators cannot be applied directly [8]. Also, *a priori* error estimators are often insufficient when encountered with solutions that contain complex flow features such as shockwaves or singularities because they rely on an asymptotic error behavior and require a regularity assumption on the solution which can cause inaccuracy in the final solution [9]. The application of such a method has been shown in a few cases such as in [10, 11, 12].

In contrast, an *a posteriori* error estimator relies on the discrete solution where the solution points are used to construct a map that provides the necessary information for the mesh adaptation algorithm to use for moving the mesh points. The benefit of such a method is that the actual solution points are used for generating the error estimator after the solution has converged, and this provides a much more direct approach to quantifying and estimating the interpolation error present in the solution. Some widely used methods for *a posteriori* error estimators include recovery techniques based on the gradient [13, 14] and approximation techniques based on piecewise interpolation of the solution nodes [4, 15] as well as its higher-order formulations [16, 17]. The foundation of this work was based on the original work of Alauzet and Loseille [18] on linear elements which introduced the concept of a ″continuous mesh framework″. The idea is to use mathematical concepts such as calculus of variation which are well defined on Riemannian metric space to construct an error estimator and use it as a tool for mesh adaptation[18, 19]. An approach using this framework for an adjoint-based error estimate for ″*hp*″ mesh adaptation can be found here [20]. Additionally, *a posteriori* error estimators can take advantage of anisotropic flow features present in the solution such as shockwave position to encode the optimal size and spacing required in the mesh to accurately resolve such flow characteristics. Shockwave positions are usually not known before solution convergence hence why an *a posteriori* error estimator is favored over *a priori* error estimators [21]. In this work, an *a posteriori* error estimator is investigated with two classes of error tracking methods which are namely the feature and adjoint-based error indicators which will be discussed shortly.

1.1.2 Riemannian Metric and Frame Fields

In the previous section, the principle of *a priori* and *a posteriori* error estimators were introduced. We shift the focus to the type of error that can be tracked for mesh adaptation and how the error estimator is built. Two common error indicators are feature and adjoint-based error indicators. The feature-based indicator is a simple way of encoding the solution error through general features of the solution such as solution gradients, or solution curvature while the adjoint-based indicator estimates the contribution of each cell to the error of a functional of interests [22]. For instance, the adjoint-based error estimator can be constructed in such a way as to track where the lift and drag coefficient errors are highest and adapt the mesh accordingly. A review paper by Alan et al. shows recent developments in error estimators for anisotropic mesh adaptation technique and presents very well the state of the art for these methods [23]. Once the error tracking approach is selected (feature or adjoint-based), this information is then encoded into a map which is known as a metric field. Riemannian metric spaces are naturally curved and are well-defined by matrices known as metrics. Their applicability to mesh adaptation has been widely employed in CFD, and the goal is to produce equidistributed edges in this space, which is generally not attainable in planar spaces as demonstrated in the work of d'Azevedo and Simpson [24, 25]. Most notably, Riemannian metrics allow for alignment of the cell with anisotropic flow features which can be controlled via the metric field. The metric field defines the size or anisotropy and the orientation of the underlying mesh topology of interest. Once the mesh is adapted through the Riemannian metric space which results in an isotropic mesh on this space, an anisotropic mesh will naturally appear in the planar space which completes the duality between the curved and planar space. In this work, quadrilateral elements are used over triangular elements in the mesh adaptation using metric fields due to their added benefit of being inherently tensor product elements. This unlocks the possibility of using sum-factorization to solve matrix-vector multiplications of an order lower than using triangles which do not have a trivial tensor product form. More information regarding quadrilateral tensor product expansions can be found in Chapters 3 and 4 of *Spectral/hp Element Methods for CFD* by Kardiadakis and Sherwin [26]. Besides the general metric field, most notably, frame fields present another interesting way of encoding the error indicator and this method arises from the field of computer graphics. This method encodes the orientation of the cell through the principal axes of the quadrilateral cell, and similarly, the length of those axes describes the size of the cell as stated by Panozzo et al. in their work [27, 28]. In addition, their work also encodes the density of cells in the frame field, which thereby facilitates their application to an anisotropic quadrilateral mesh generator.

1.1.3 Metric-based Mesh Adaptation

Early work on two-dimensional grid refinement based on the truncation error of the solution was attempted by Beger and Jameson [29] and also, Beger and Oliger [30]. They showed the benefit of using local grid refinement on quadrilateral meshes for solving hyperbolic problems and Euler's equation at a fraction of the computational cost compared to a uniform grid. The refinement process works by splitting a single cell by four for every cell that is flagged during the grid refinement process. However, one drawback of such methods using local refinement is the reliance on the initial grid since the grid can only converge with refinement based on the initial location of the grid points. Additionally, certain flow features of dominance such as shockwaves or boundary layers result in high-gradient regions that can be overlooked by the refinement process, leaving less apparent features badly resolved in the final solution [31].

To alleviate this issue, more work began to focus on error estimators that are based on the direction features of the flow rather than the change in the high-gradient region of the solution. This in part contributed to the beginning of anisotropic mesh adaptation where the anisotropic feature of the flow is prescribed into a metric acting as an error estimator. Peraire et al. were the first to attempt such methods on two-dimensional triangular linear grids where the mesh would align with directional features from the solution [32]. Their method involved the reconstruction of the Hessian of the solution based on the density of the solution as an error indicator. As a result, directional error estimates inherently surfaced from the Hessian matrix and the mesh was able to adapt based on the directional properties of the flow. A three-dimensional version of this work has been introduced shortly after which can be found in this paper [33]. Then, in regards to encoding the error estimator inside a Riemmanian metric field, the first to introduce and attempt mesh adaptation based on this concept was George et al. [34]. Their work focused on generating an isotropic mesh (i.e., a mesh where the edge lengths are uniform) using the Hessian in the Riemmanian space, and this resulted in an anisotropic mesh in the physical space. Later, adjoint-based mesh adaptation started to gain popularity due to its capability of adapting the mesh based on a functional of interest, and Becker and Rannachar introduced the concept of dual-weight residual [35]. Venditi and Darmofal performed pioneering work in applying this concept to anisotropic mesh adaptation which includes inviscid and viscous flow [36, 37, 38]. More recently, high-order works on this subject has also been attempted by Dolejší [16, 17]. Lastly, MacLean and Nadarajah introduced an anisotropic mesh adaptation based on the L_p-CVT method which adapts to both feature- and adjoint-based error estimators [39]. In this work, we will attempt to extend it by including boundary treatments for the L_p-CVT. The goal is to investigate the applicability of the boundary modification and also, the benefit that it brings to tracking the anisotropic features more effectively at the boundary.

1.2 Thesis Overview

This thesis is structured as follows. First, Chapter 2 will present a broad overview of mesh generation and adaptation techniques. Specifically, the generation of quadrilateral structured mesh will be covered for both two-dimensional and three-dimensional space to provide a simple introduction to mesh generation and how it applies to the field of mesh adaptation. Additionally, the L_p-CVT mesh generator will be explained in more detail to show how boundary treatments are utilized to better depict optimal mesh points at boundaries. Second, in Chapter 3, we present a review of the discontinuous Galerkin (DG) method for the discretization of conservation laws. Third, Chapter 4 will show an overview of the continuous mesh models and describe the relationship between the L_p-CVT mesh adaptation technique and the DG method. Then, in Chapter 5 we will present the numerical results comparing both with and without boundary treatment mesh adaptation on analytical and discrete frame fields. In particular, an applied test case using the L_p-CVT mesh generator will be shown for solving a transonic flow around a NACA0012 airfoil. Finally, Chapter 6 will conclude the results and present future works that remain to be investigated.

Chapter 2

Mesh Generation

Mesh generation constitutes the initial phase preceding the resolution of any computational (numerical) problem. The choice of a numerical approach for solving governing equations necessitates the discretization of the domain of interest to compute solutions at various mesh points. Additionally, the mesh serves not only as a tool for numerical computation but also as a means of visually representing flow features which helps communicate the solution to the end-users. In contrast to solvers, mesh generation involves numerous constraints that are not solely user-dependent, but also include considerations tied to the evolving solution itself. This enforces the significance of mesh adaptation which is a technique frequently employed to minimize user intervention in mesh generation while also enhancing the resolution of the solution in each adaptation cycle. All in all, this chapter aims to provide a comprehensive overview of mesh generation, introducing key principles and techniques of both mesh generation and adaptation.

2.1 Structured vs Unstructured Mesh

In the context of this work, where two-dimensional domains are of primary interest, grids can be classified into two main types: structured and unstructured meshes. A structured mesh is highly organized, presenting a patterned grid that enables the solver to locate specific grid points using a unique set of indices along the grid lines, as illustrated in Figure 2.1. On the other hand, unstructured grids lack any sense of structure and are inherently disorganized. While this allows for flexibility in representing arbitrary domains by conforming to the shape of the geometry without requiring a predefined structure, it comes at the cost of requiring specific data structures for retrieving their locations.

A straightforward technique for generating an unstructured mesh based on triangular elements is the Delaunay Triangulation (DT) which is discussed in Section 2.3.1. An example of a basic implementation of this technique is presented by Persson et al. [40], where the author employs DT to generate a mesh and optimizes mesh points using a force-based smoothing procedure. The mesh boundary is defined by a signed distance function (SDF), distinguishing nodes inside or outside a specified boundary of interest. Figure 2.2, produced by DistMesh, illustrates this process.

Alternative techniques such as Constrained Delaunay Triangulations (CDT) can be employed as found in [41] which forms the principle behind the popular mesh generation software GMSH [42]. This approach involves adding a boundary constraint to DT, allowing for mesh generation within a specified domain defined by the boundary constraint (e.g., a square or circular domain, as depicted in Figure 2.2). In addition to defining the shape of the boundary, the distribution of the points along the boundary can also be specified. For instance, Figure 2.1 shows an equidistant distribution of points along the x-axis (horizontal boundary) and a 2^{nd} order polynomial growth function along the y-axis (vertical boundary).

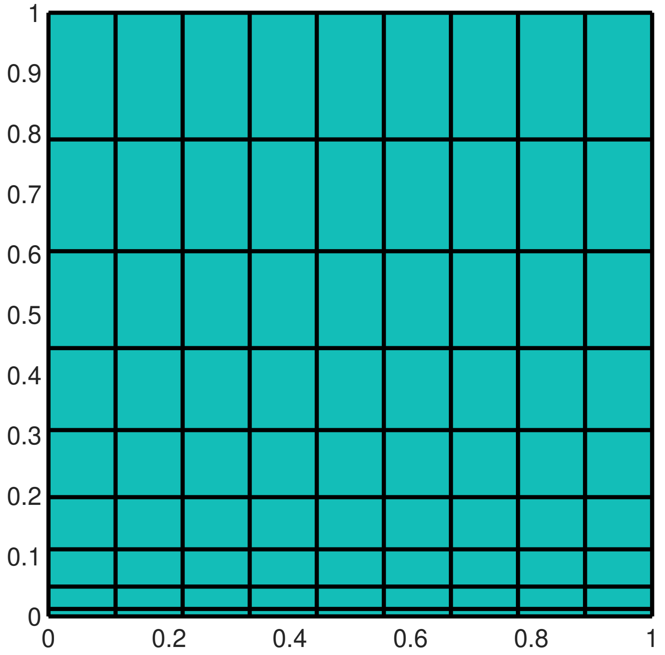


Figure 2.1: Two-Dimensional Structured Grid with 2^{nd} order polynomial growth function along the y-axis.

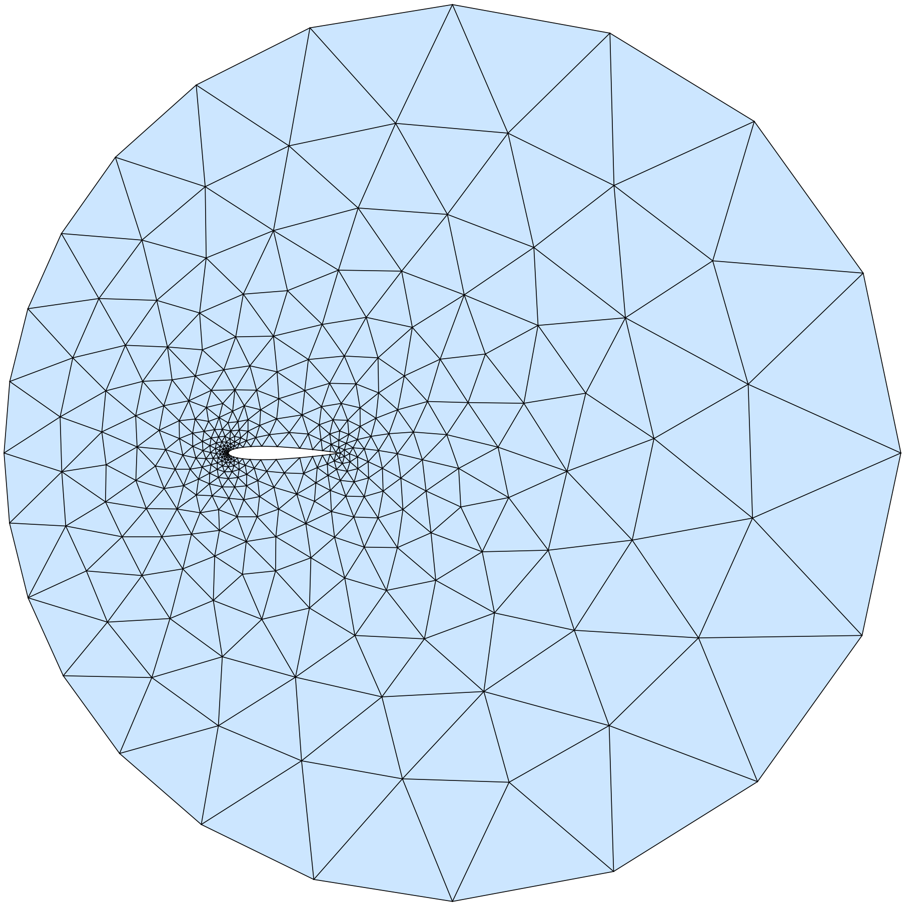


Figure 2.2: Two-Dimensional Delaunay Triangulation over a NACA0012 airfoil bounded by a circular domain produced by DistMesh.

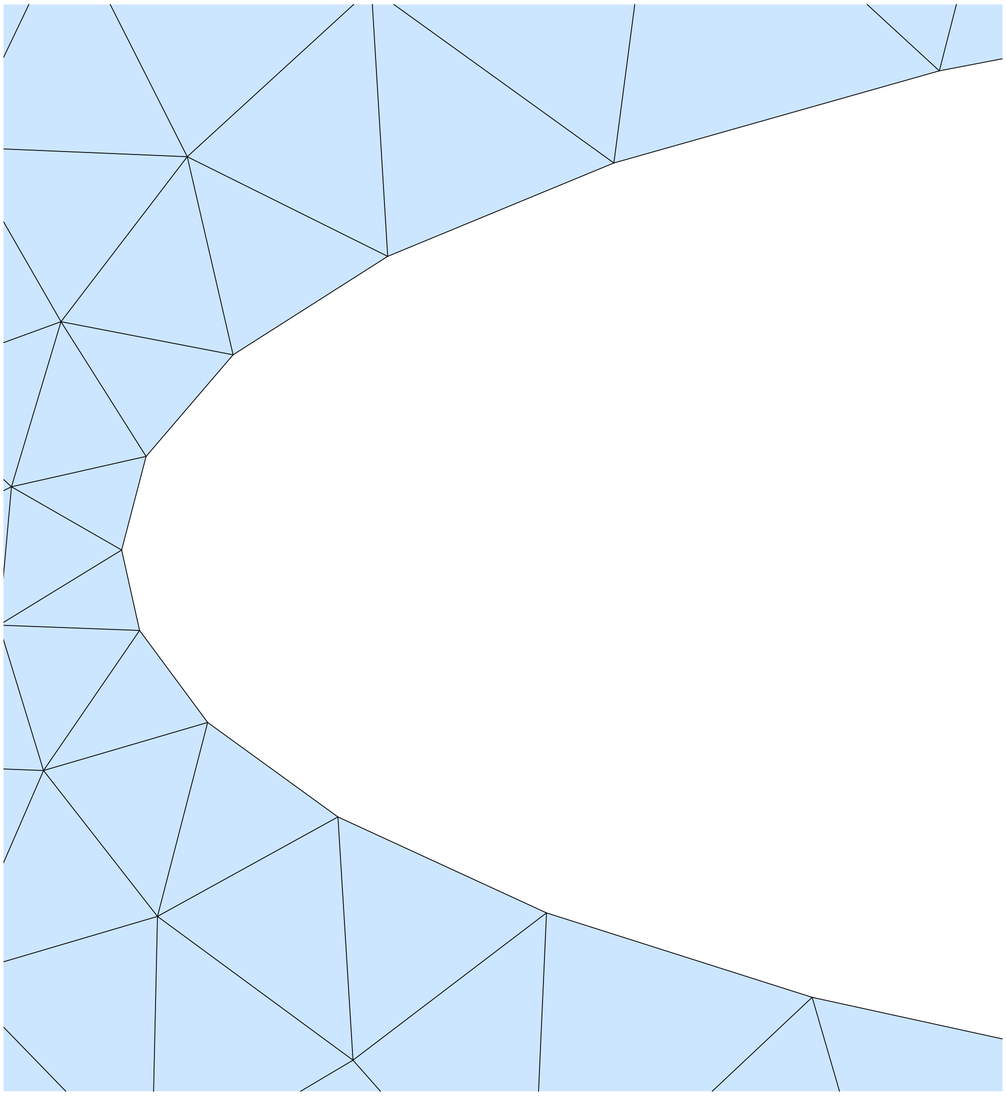


Figure 2.3: Leading Edge of NACA0012 airfoil bounded by a circular domain produced by DistMesh.

In Figure 2.1, an equidistant point distribution along the x-axis, coupled with a polynomial function ($y = x^2$), is employed to determine the spacing along the y-axis. Due to the point distribution on the boundary, comprising 40 points in both the x and y directions, the intersection of horizontal and vertical lines creates individual mesh points. The combination of mesh points generates the computational domain.

Figure 2.2 shows an unstructured grid with triangular elements around a NACA0012 airfoil inside a circular domain and a close-up view of the leading edge of the airfoil is shown in Figure 2.3. In this case, unstructured mesh proves effective in arbitrary domains where triangular elements are well-fitted inside the domain conforming to both the airfoil and circular boundary. Interestingly, the airfoil's leading edge is discretized with linear elements (i.e., straight lines) which consequently penalizes the depiction of the leading edge curvature. The resolution of the leading edge curvature is therefore influenced by the chosen number of points which is typically user-defined. Despite the apparent simplicity of mesh generation, achieved by inputting spacing distribution on each axis using a chosen function (e.g., exponential function, inverse hyperbolic tangent function, etc.), numerical methods and mesh generation, in general, are not as straightforward as they may seem. The earlier example illustrated a straightforward geometry, but mesh generation complexity arises when dealing with intricate geometries, particularly those involving curved boundaries (as illustrated in Figure 2.2). Simple straight lines are inadequate for satisfying results in the presence of curved boundaries due to a lack of resolution when too few elements are used. Furthermore, specific grid constraints, such as orthogonality on the initial grid layer adjacent to boundaries, may be necessary which adds complication to the mesh generation process.

2.2 Computational vs Physical Domain

In mesh generation and numerical methods, establishing a reference system is often crucial. There exist two distinct domains which are namely the computational and physical domains that play a pivotal role. The computational domain is characterized by straight-sided boundaries with spacing along the x-axis and y-axis that are equidistant, eliminating the need for specifying any customized spacing. This often facilitates the computations performed by the solver by disregarding any numerical schemes coupled with non-equidistant spacing. Conversely, the physical domain is typically defined by the geometry of the domain of interest, often with prescribed boundary curvature and point distribution. It is crucial to recognize that both domains coexist due to a one-to-one relationship between each grid point located in both domains [43]. The following example illustrates both domains, as presented in [43].

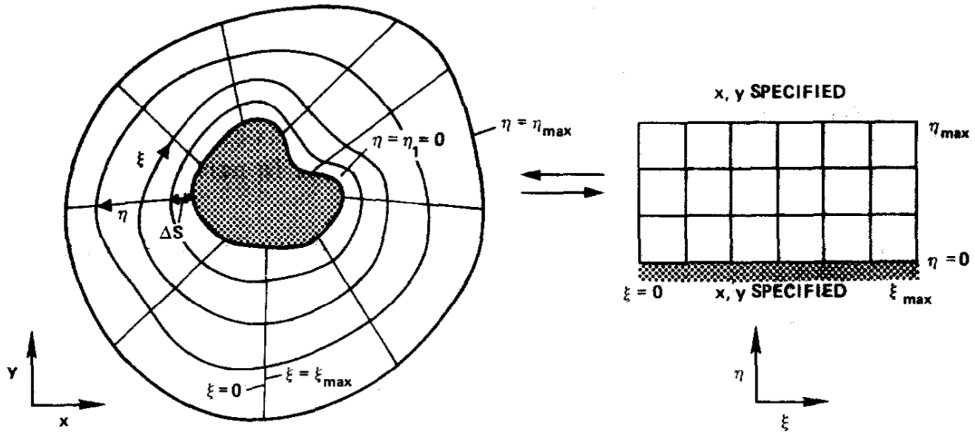


Figure 2.4: Left: Physical Domain — Right: Computational Domain [43]

2.2.1 Transformation Matrix

The transformation matrices define the conversion between the physical and the computational domain and are shown as follows,

$$\begin{bmatrix} \xi \\ \\ \eta \end{bmatrix} = \begin{bmatrix} \xi(x,y) \\ \\ \eta(x,y) \end{bmatrix} \qquad\qquad \boldsymbol{J_1} = \begin{bmatrix} \xi_x & \xi_y \\ \\ \eta_x & \eta_y \end{bmatrix}$$

$$\begin{bmatrix} x \\ \\ y \end{bmatrix} = \begin{bmatrix} x(\xi,\eta) \\ \\ y(\xi,\eta) \end{bmatrix}, \qquad \boldsymbol{J_2} = \begin{bmatrix} x_\xi & x_\eta \\ \\ y_\xi & y_\eta \end{bmatrix},$$

$$\boldsymbol{J_1} = [\boldsymbol{J_2}]^{-1}, \tag{2.1}$$

$$J = det[\boldsymbol{J_2}] = x_\xi y_\eta - x_\eta y_\xi, \tag{2.2}$$

where ξ denotes the x variable in the computational domain, η denotes the y variable in the computational domain, and finally, x and y are the same definitions as from the physical domain. $\boldsymbol{J_1}$ is the Jacobian matrix for the computational domain variables, $\boldsymbol{J_2}$ is the Jacobian matrix for the physical domain variables, and finally, J is the Jacobian determinant which is defined as the determinant of $\boldsymbol{J_2}$.

The derivatives that are shown previously can be obtained through finite differences. Typically, a central finite difference is used for inner grids, whereas forward and backward finite differences are used instead at the boundaries. Depending on the location of the grid points, forward difference can be used at the first boundary line and backward difference at the last boundary line for both ξ and η directions.

To convert between both domains, the following relationships can be used,

$$f_x = \frac{\partial(f,y)}{\partial(\xi,\eta)} \Big/ \frac{\partial(x,y)}{\partial(\xi,\eta)} = \frac{y_\eta f_\xi - y_\xi f_\eta}{J}. \tag{2.3}$$

$$f_y = \frac{\partial(x,f)}{\partial(\xi,\eta)} \Big/ \frac{\partial(x,y)}{\partial(\xi,\eta)} = \frac{-x_\eta f_\xi + x_\xi f_\eta}{J}. \tag{2.4}$$

One might question the need for going through the process of obtaining derivative values of a function of the ξ and η directions. The answer lies in the fact that the earlier established relationships enable the formulation of equidistant finite differences, as mentioned previously. It is interesting to note that all the derivatives on the right-hand side of equation 2.3 and 2.4 depend on the computational domain variables, making them easily computable. This ease arises from the uniform distribution of spatial spacing between grid lines in both the ξ and η directions, rendering $\triangle\xi$ and $\triangle\eta$ constants.

In the physical domain, challenges may arise if the grid is clustered towards $\xi = 1$, as shown in the case of a C-Mesh illustrated in Figure 2.5, where grid lines are concentrated near the airfoil. In such instances, the use of non-equidistant finite differences for solution computation becomes necessary, introducing complexity to the process. On the contrary, this is eliminated through performing computations in the computational domain which ensures a clean and efficient computational environment for the solver. This thereby facilitates easy tracking of grid points and their locations at the expense of a conversion step through equation 2.3 and 2.4.

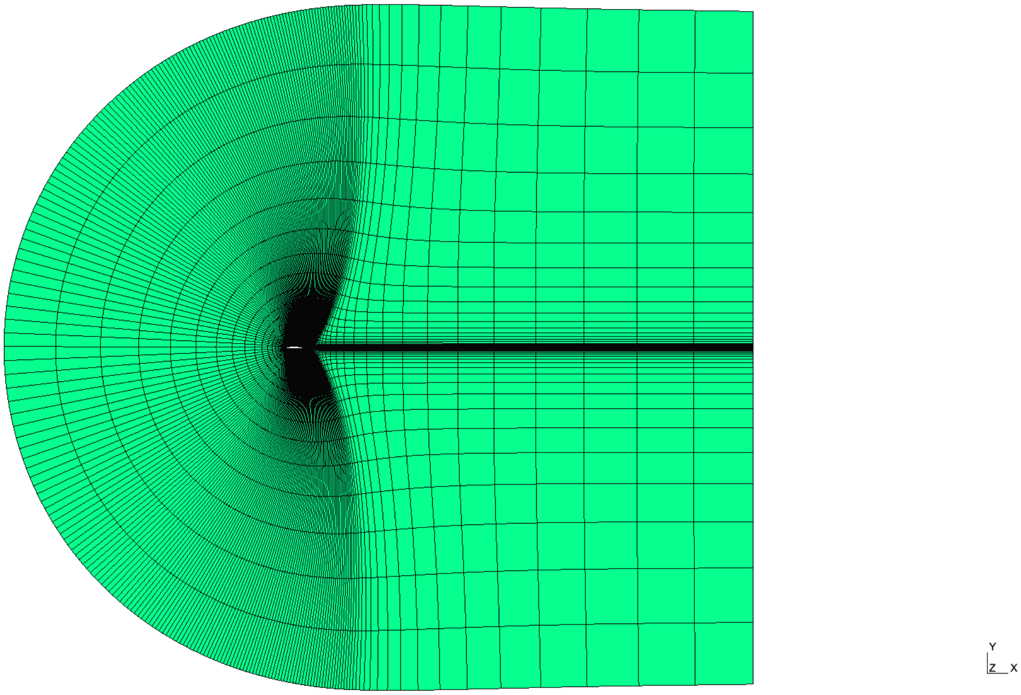


Figure 2.5: Structured CMesh Grid with quadrilateral cells generated using a Hyperbolic Grid Generator

2.3 Two-Dimensional Triangular Mesh Generator

In this section, a brief overview of the Delaunay Triangulation as well as the Voronoi diagram will be discussed. This will form the basis of the L_p-CVT method shown in section 2.8.

2.3.1 Delaunay Triangulation

The Delaunay Triangulation (DT) stands out as a widely employed technique in mesh generation, generating triangles in two-dimensional and tetrahedra in three-dimensional space. The approach involves connecting nodes within a domain to form triangular elements, ensuring that no points reside within the circumcenter of the triangles known as the ″empty circumdisk property″ condition, originally introduced by Delaunay in 1934 [44]. This condition serves to maximize the minimum angle within any triangle in the triangulation, leading to an optimal mesh [45]. Various forms of optimal DT based on diverse criteria have been explored over time. For instance, optimal DT can be generated by minimizing the linear interpolation error for a given function [46] or by employing constrained optimization based on predefined boundary edges [41]. The latter case, known as Constrained Delaunay Triangulation (CDT), will be the subject of investigation in this study. Figure 2.6 illustrates an example produced by DistMesh, showcasing a CDT generated with a square outer boundary and a circular cavity within the domain.

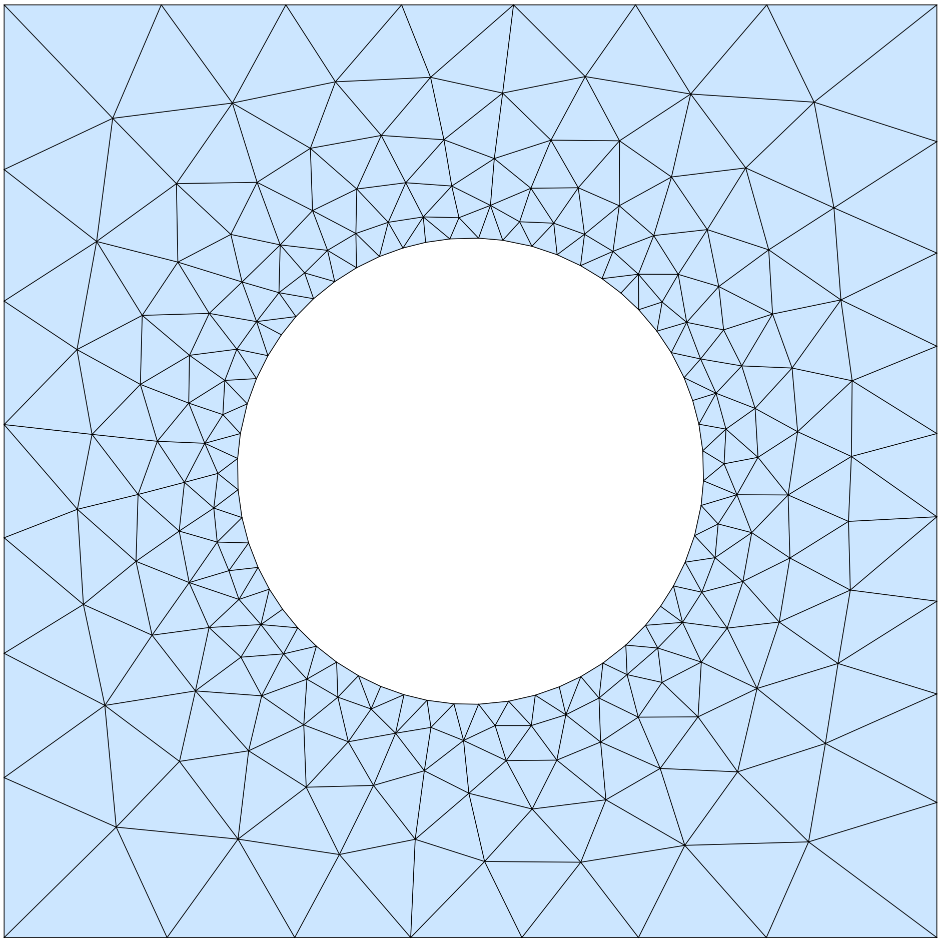


Figure 2.6: CDT generated by DistMesh inside a bounded domain

One of the major properties of DT is that it has a geometric dual called a Voronoi Diagram (VD) [47]. This duality is shown in Figure 2.7 where the red-colored map is the Voronoi Diagram and the black-colored map is the DT superimposed on the Voronoi Diagram. The idea is to connect all the Voronoi sites to form triangles. Three Voronoi sites (i.e. $\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3$) would be used for generating one triangle $\mathcal{T}_1$. This will form the basis of the L_p-CVT method which is based on these two concepts as presented in the section 2.8.

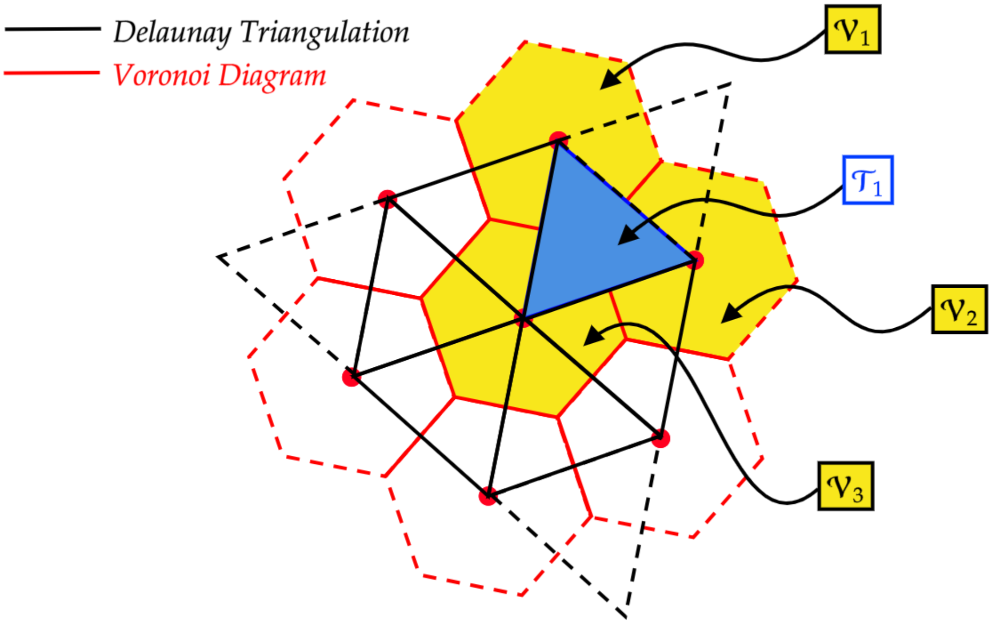


Figure 2.7: Geometric duality between Delaunay Triangulation and Voronoi Diagram

2.3.2 Voronoi Diagram

VD is built based on the premise that any point, p, inside a Voronoi cell, $\mathcal{V}_i$, is the closest to its Voronoi site, P_i, compared to any other Voronoi sites, P_j, present in the domain. This forms a set of non-overlapping convex polygons which are formulated as follows [47],

$$\{\mathcal{V}_i\} = \{p : \|p - P_i\| \ < \|p - P_j\|, \forall i \neq j\} \tag{2.5}$$

One special type of VD that is of interest in this work is called the Centroidal Voronoi Diagram (CVD). This type of VD is very similar to the definition of 2.5, however, the Voronoi sites, P_i, are optimized to be aligned with the centroid of the Voronoi cell. Figure 2.8 shows a comparison between a regular Voronoi cell and a Centroidal Voronoi cell. This is formulated as an optimization problem where the functional of interest is written as follows from [48],

$$F(\boldsymbol{x}) = \sum_i \int_{\Omega_i \in \boldsymbol{\Omega}} \|\boldsymbol{y} - \boldsymbol{x_i}\|^2 d\boldsymbol{y}, \tag{2.6}$$

where Ω_i is the corresponding Voronoi cell, $\boldsymbol{\Omega}$ is the domain of interest, $\boldsymbol{y_i}$ is any point located inside or at the boundary of the Voronoi cell, Ω_i, and $\boldsymbol{x_i}$ is the corresponding Voronoi site of Ω_i. Since F is of class C^2 continuous as shown in [49], this unlocks the possibility of using Newton's method to optimize F. The gradient of equation 2.6 is written as follows from [50] which can be employed with a quasi-Newton BFGS (Broyden–Fletcher–Goldfarb–Shanno) method to solve the minimization of equation 2.6,

$$\nabla F|_{\boldsymbol{x_i}}(\boldsymbol{x_i}) = 2m_i(\boldsymbol{x_i} - \boldsymbol{g_i}), \tag{2.7}$$

where m_i is the volume of the Voronoi cell, and g_i, is the centroid of the Voronoi cell.

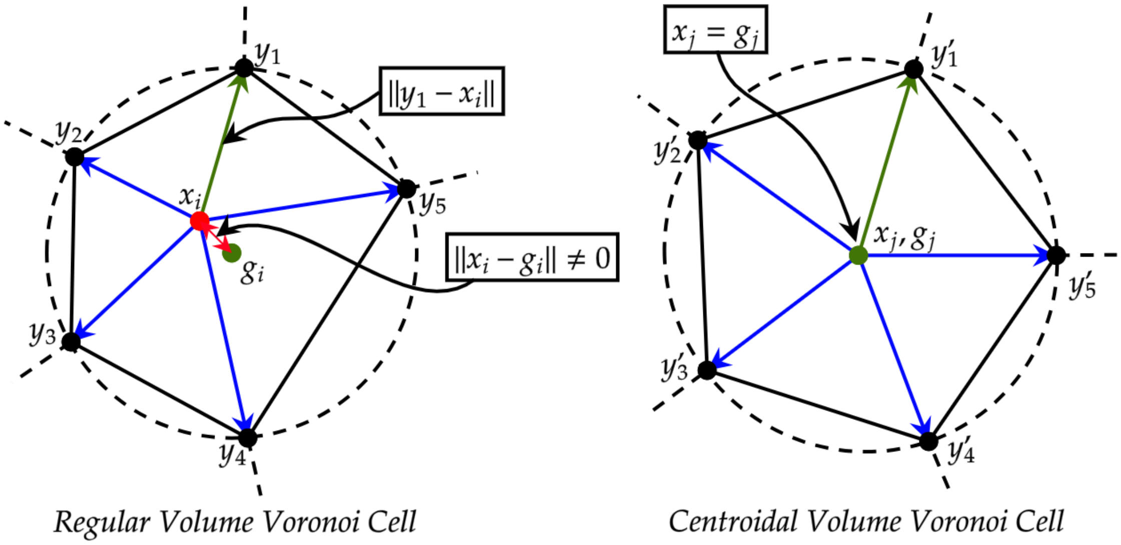


Figure 2.8: Comparison of regular Voronoi cell and Centroidal Voronoi cell

2.4 Two-Dimensional Algebraic and PDEs based Quadrilateral Structured Mesh Generator

Various methods exist for generating two-dimensional meshes. Amongst the most popular are algebraic methods [51, 52], the partial differential equation (PDE) approach [43, 53, 54, 55], conformal mapping [56], and other methods [57]. This section will focus on algebraic methods and the application of the partial differential equation approach.

As a supplementary note, algebraic methods hold a personal preference due to their simplicity and rapid computational speed. However, it is crucial to acknowledge their inherent limitations, which can significantly diminish their practicality. While algebraic methods serve as an excellent starting point for learning about meshing, they may not be the optimal choice for consistently generating high-quality meshes in the long run. Within the field of purely algebraic methods, noteworthy techniques include Transfinite Interpolation and Hermite Interpolation (in addition to other methods, which are beyond the scope of this section). The appeal of these methods lies in their computational efficiency, as they dynamically compute the grid, allowing for near-instantaneous results within a single loop. Notably, these methods operate without the need for iterative convergence to address errors, as they follow a purely mathematical approach where inputting a value into a function yields an output grid.

2.4.1 Transfinite Interpolation

Transfinite Interpolation is the most basic approach to generating a two-dimensional grid. Essentially, this approach employs linear interpolation, where the grid lines are interpolations from the boundaries. Consequently, the closer a grid line is to the boundary, the more it mirrors the boundary. The equation used for computing the transfinite grid is noted as follows from [58],

$$\begin{aligned} x(\xi,\eta) = {} & x(\xi_{max},\eta)\xi + (1-\xi)x(0,\eta) + x(\xi,\eta_{max})\eta + (1-\eta)x(\xi,0) \\ & - \xi\eta x(\xi_{max},\eta_{max}) - \xi(1-\eta)x(\xi_{max},0) - (1-\xi)\eta x(0,\eta_{max}) \\ & - (1-\xi)(1-\eta)x(0,0) \end{aligned} \tag{2.8}$$

$$\begin{aligned} y(\xi,\eta) = {} & y(\xi_{max},\eta)\xi + (1-\xi)y(0,\eta) + y(\xi,\eta_{max})\eta + (1-\eta)y(\xi,0) \\ & - \xi\eta y(\xi_{max},\eta_{max}) - \xi(1-\eta)y(\xi_{max},0) - (1-\xi)\eta y(0,\eta_{max}) \\ & - (1-\xi)(1-\eta)y(0,0) \end{aligned} \tag{2.9}$$

One key observation about the formula though is that the ξ and η are defined as equidistant step lengths in each direction. This approach enables the expression of the computational domain as a unit spatial domain with *N* x *N* grid points. Figure 2.9 shows an example of a transfinite mesh. The blue points are boundary points that are predefined in the physical domain which defines the shape of the boundary.

Clearly, in the vicinity of the semicircle, the grid points are more clustered and conform to the curvature of the circular boundary. Conversely, upon closer examination of the grid lines near the upper region, their trajectory appears nearly linear, aligning with the straight-sided upper boundary. Note that in transfinite interpolation, since the grid lines inherently conform to the boundaries, it is possible to identify different sections in the mesh based on the shape of the boundary. From Figure 2.9, three well-defined sections emerge, seperated at $x = 0.25$ and $x = 0.75$. This outcome is expected due to the prescribed function along the bottom boundary in the ξ direction. This function entails a linear segment, succeeded by a semicircular segment, and concludes with another linear segment. Consequently, the execution of the transfinite algorithm results in a mesh that embodies these three predetermined sections, as illustrated in Figure 2.9.

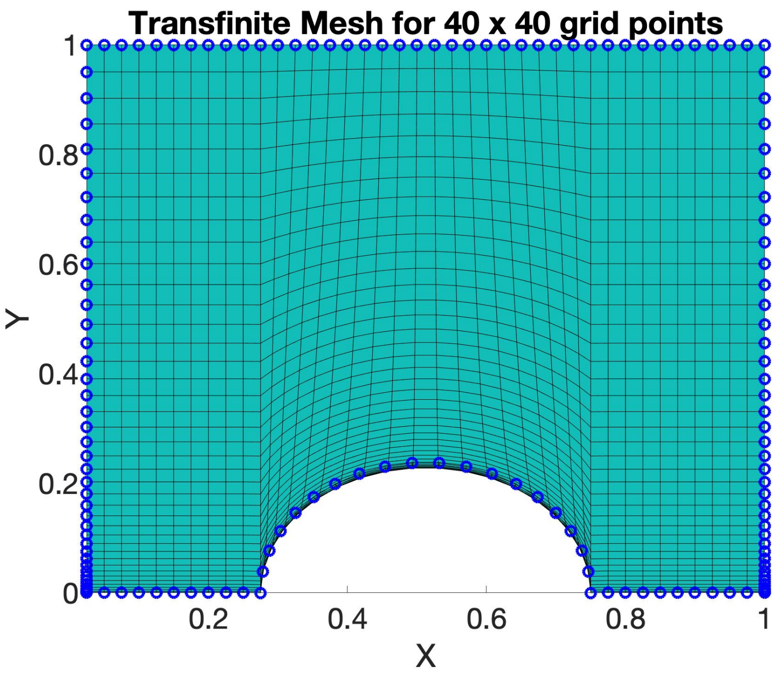


Figure 2.9: Transfinite Interpolation of a rectangle with a semicircle.

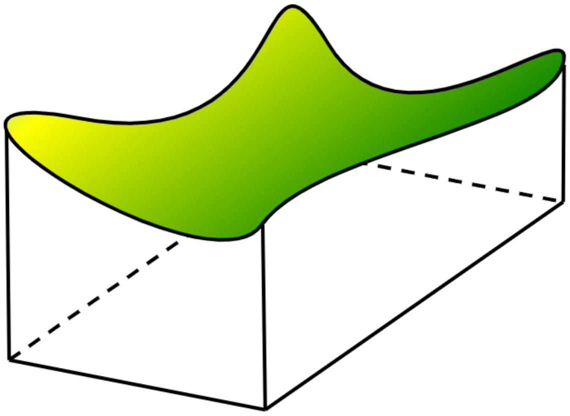


Figure 2.10: Transfinite Interpolation in three dimensional space as demonstrated in [58]

2.4.2 Hermite Interpolation

In Hermite Interpolation, a distinct difference from Transfinite Interpolation is the requirement of derivatives at the boundary. At first glance, this method seems to add control over the grid, allowing for the specification of parameters to shape the grid's appearance at the boundary. However, it is not always so obvious how to pick the first layer of grid lines to run the Hermite Interpolation since there are many ways to determine the first adjacent grid line to the boundary.

Selecting the first grid lines refers to deciding the first inner grid lines for first-order finite difference at the boundaries which is required in the formula for the Hermite Interpolation. One way to define them is to make the first inner grids orthogonal to the boundaries, however, this may not always be the case since not all grids require this property. Additionally, the inner grid points are free to move and hence, keeping the first layer orthogonal may not yield good results as it could interfere with the solution and provide non-smooth solutions.

The formula for the Quadratic Hermite Interpolation is as follows,

$$\begin{aligned}
x_{i,j} = {} & \alpha_0(\xi)x_{0,j} + \alpha_0'(\xi)(x_\xi)_{0,j} + \alpha_1(\xi)x_{i_{max},j} + \alpha_1'(\xi)(x_\xi)_{i_{max},j} \\
& + \beta_0(\eta)x_{i,0} + \beta_0'(\eta)(x_\eta)_{i,0} + \beta_1(\eta)x_{i,j_{max}} + \beta_1'(\eta)(x_\eta)_{i,j_{max}} \\
& - [\alpha_0(\xi)\beta_0(\eta)x_{0,0} + \alpha_0(\xi)\beta_0'(\eta)(x_\eta)_{0,0} \\
& + \alpha_0'(\xi)\beta_0(\eta)(x_\xi)_{0,0} + \alpha_0'\beta_0'(\eta)(x_{\eta\xi})_{0,0} \\
& + \alpha_0(\xi)\beta_1(\eta)x_{0,j_{max}} + \alpha_0(\xi)\beta_1'(\eta)(x_\eta)_{0,j_{max}} \\
& + \alpha_0'(\xi)\beta_1(\eta)(x_\xi)_{0,j_{max}} + \alpha_0'(\xi)\beta_1'(\eta)(x_{\xi\eta})_{0,j_{max}} \\
& + \alpha_1(\xi)\beta_0(\eta)x_{i_{max},0} + \alpha_1(\xi)\beta_0'(\eta)(x_\eta)_{i_{max},0} \\
& + \alpha_1'(\xi)\beta_0(\eta)(x_\xi)_{i_{max},0} + \alpha_1'(\xi)\beta_0'(\eta)(x_{\xi\eta})_{i_{max},0} \\
& + \alpha_1(\xi)\beta_1(\eta)x_{i_{max},j_{max}} + \alpha_1(\xi)\beta_1'(\eta)(x_\eta)_{i_{max},j_{max}} \\
& + \alpha_1'(\xi)\beta_1(\eta)(x_\xi)_{i_{max},j_{max}} + \alpha_1'(\xi)\beta_1'(\eta)(x_{\xi\eta})_{i_{max},j_{max}}]
\end{aligned} \tag{2.10}$$

$$\alpha_0(\xi) = 2\xi^3 - 3\xi^2 + 1, \;\; \alpha_0'(\xi) = \xi^3 - 2\xi^2 + \xi \tag{2.11}$$

$$\alpha_1(\xi) = -2\xi^3 + 3\xi^2, \;\; \alpha_1'(\xi) = \xi^3 - \xi^2 \tag{2.12}$$

$$\beta_0(\eta) = 2\eta^3 - 3\eta^2 + 1, \;\; \beta_0'(\eta) = \eta^3 - 2\eta^2 + \eta \tag{2.13}$$

$$\beta_1(\eta) = -2\eta^3 + 3\eta^2, \;\; \beta_1'(\eta) = \eta^3 - \eta^2 \tag{2.14}$$

These equations are taken from [59]. Note that the previous equations apply for the y coordinate as well by simply exchanging x for y. From this equation, Hermite Interpolation does show more complication compared to Transfinite Interpolation since it requires the definition of both the first and second-order derivatives at the boundaries. In particular, the second-order partial derivative requires the corner points for its computation (using central finite difference) which thereby adds difficulty in the calculation. For the Cubic Hermite Interpolation, it is shown as follows,

$$x(\xi,\eta) = (1-3\eta^2+2\eta^3)x(\xi,0)+\eta^2(3-2\eta)x(\xi,1)+\eta(1-\eta)^2\frac{\partial x}{\partial \eta}(\xi,0)+\eta^2(\eta-1)\frac{\partial x}{\partial \eta}(\xi,1) \tag{2.15}$$

Note that the previous equations apply for the y coordinate as well by simply exchanging x for y. In the cubic equation, the derivatives are required at the first boundary and are only of first-order derivatives, thus, implementing this approach is much easier. Equation 2.15 is taken from [57].

2.4.3 Parabolic Equation Approach

In contrast to previous grid generation methods, PDE approaches are less straightforward. They require a solution to the governing partial equation of choice for the output mesh to be generated. The solution is the grid itself (i.e. the position of x and y coordinates). Parabolic equation approaches are inherently diffusive as they include a second-order derivative and as a consequence, boundary discontinuity is often prevented. The parabolic system is shown as follows,

$$\frac{\partial x}{\partial \eta} - A\frac{\partial^2 x}{\partial \xi^2} = S_x \tag{2.16}$$

$$\frac{\partial y}{\partial \eta} - A\frac{\partial^2 y}{\partial \xi^2} = S_y \tag{2.17}$$

The typical approach to solving this PDE would be to first generate the boundaries of the grid inside the physical domain. Then, the solver would march from the first boundary layer in the vertical direction, η = 1, up to the last layer, η = NY.

2.4.4 Elliptic Equation Approach

For the elliptic equation approach, two popular governing equations are of interest and they are known as the Laplace and Poisson equation. For instance, the heat equation for a steady-state two-dimensional problem can be reduced to an elliptical PDE. The problem can then be solved using already well-known numerical methods with corresponding boundary conditions. Boundary conditions would then be required for initializing the problem by specifying the temperature of the boundaries which is analogous to specifying the grid points on the boundary for mesh generation purposes. The solution would result in the following from Figure 2.11,

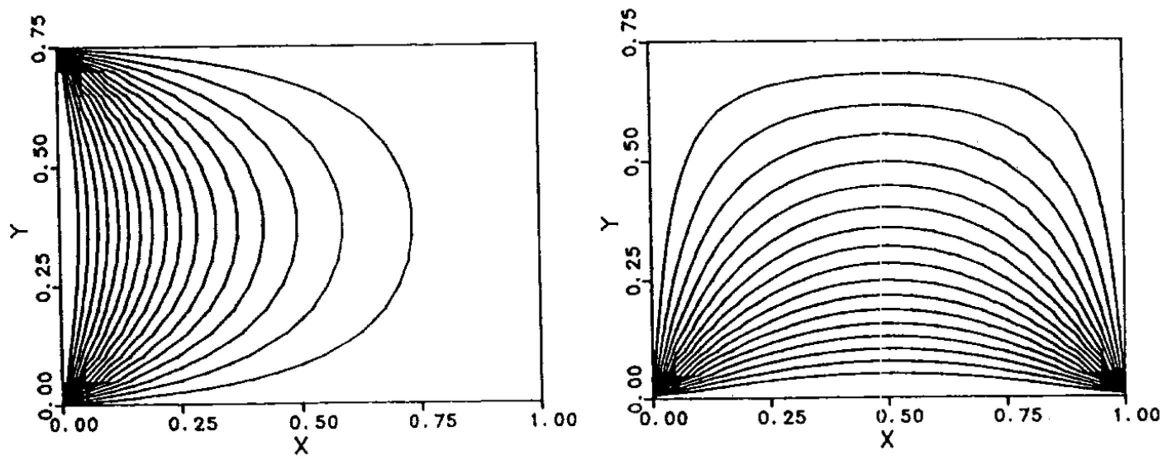


Figure 9-20. Isothermal lines for the two rectangular domains.

Figure 2.11: Solution to Heat Equation - Isothermal lines [60]

Once the solution isothermal lines are obtained, it is also possible to superimpose the isolines together to yield the following from Figure 2.14,

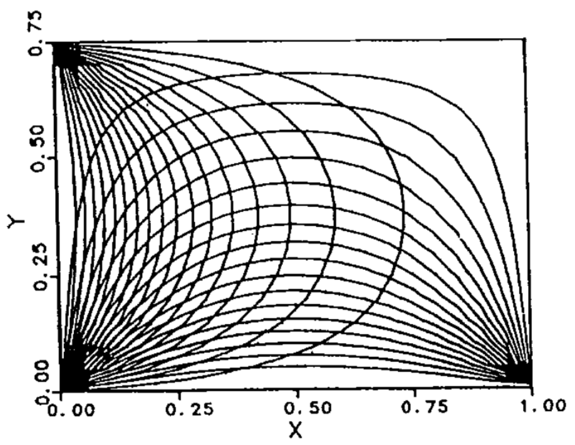


Figure 9-21. The superimposed solution for the rectangular domain.

Figure 2.12: Superimposed solution of Heat Equation[60]

From Figure 2.14, it is noticeable that the overlapping isothermal lines from both solutions of Figure 2.11 are similar to grid points. This is the logic behind the elliptic equation approach where the goal is to solve the elliptical PDE to then generate a mesh by superimposing the isothermal lines that are controlled using the boundary condition. In the case of the Poisson equation, a forcing function can also be used for additional effects such as attraction and repulsion of points. The output mesh will then benefit from the forcing function by adding possible customization to alter the isothermal lines in a way that the end user will be able to define a grid of interest that is suitable for his or her test case.

The elliptical PDE is presented as follows for the Laplace and Poisson equations. Note that the Laplace equation is derived from the Poisson, but with the forcing function removed.

$$\xi_{xx} + \xi_{yy} = P(\xi, \eta) \tag{2.18}$$

$$\eta_{xx} + \eta_{yy} = Q(\xi, \eta) \tag{2.19}$$

Using the following relationships derived from the transformation between the Physical and Computational domains in section 2.2.1, the elliptical PDE can be converted into the physical domain by using,

$$\xi_x = y_\eta / J \tag{2.20}$$

$$\xi_y = -x_\eta / J \tag{2.21}$$

$$\eta_x = -y_\xi / J \tag{2.22}$$

$$\eta_y = x_\xi / J \tag{2.23}$$

$$J = x_\xi y_\eta - x_\eta y_\xi . \tag{2.24}$$

Now, the elliptic PDE is written as follows,

$$\alpha x_{\xi\xi} - 2\beta x_{\eta\xi} + \gamma x_{\eta\eta} = -J^2(P x_\xi + Q x_\eta) \tag{2.25}$$

$$\alpha y_{\xi\xi} - 2\beta y_{\eta\xi} + \gamma y_{\eta\eta} = -J^2(P y_\xi + Q y_\eta), \tag{2.26}$$

where,

$$\alpha = x_\eta^2 + y_\eta^2 \tag{2.27}$$

$$\beta = x_\xi x_\eta + y_\xi y_\eta \tag{2.28}$$

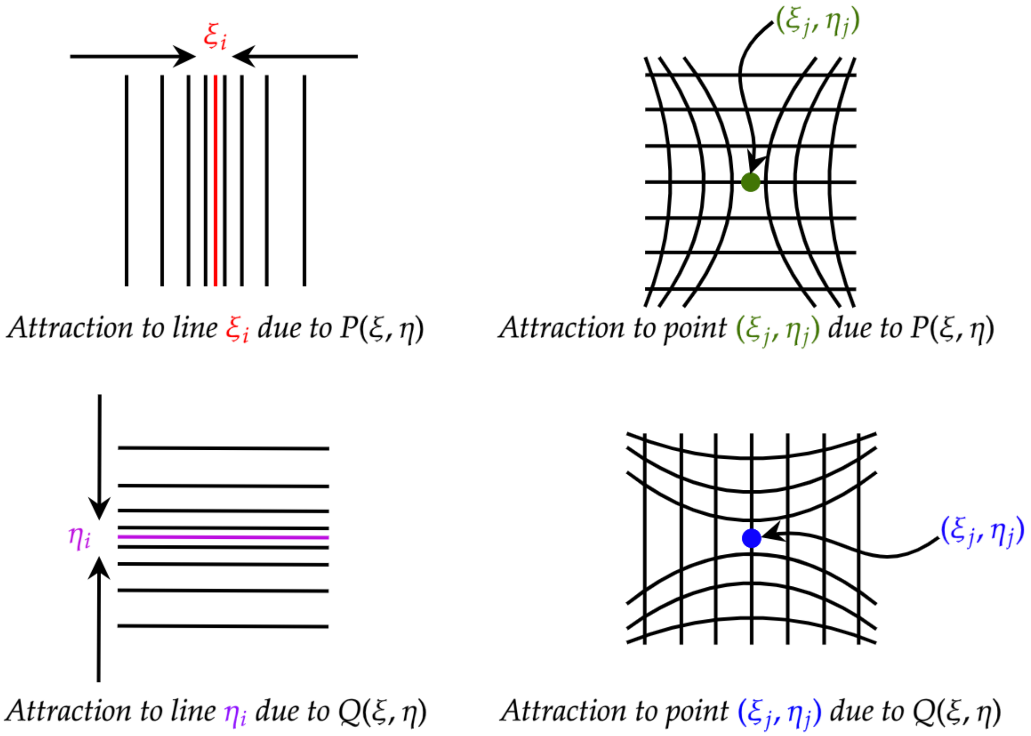
$$\gamma = x_\xi^2 + y_\xi^2 \tag{2.29}$$

The forcing functions are defined as follows,

$$\begin{aligned} P(\xi,\eta) = &-\sum_{i=1}^{n} a_i sgn(\xi - \xi_i) exp\left[-c_i|\xi - \xi_i|\right] \\ &-\sum_{j=1}^{m} b_j sgn(\xi - \xi_j) exp\left[-d_j\sqrt{(\xi-\xi_j)^2 + (\eta-\eta_j)^2}\right] \end{aligned} \tag{2.30}$$

$$\begin{aligned} Q(\xi,\eta) = &-\sum_{i=1}^{n} a_i sgn(\eta-\eta_i) exp\left[-c_i|\eta-\eta_i|\right] \\ &-\sum_{j=1}^{m} b_j sgn(\eta-\eta_j) exp\left[-d_j\sqrt{(\xi-\xi_j)^2+(\eta-\eta_j)^2}\right] \end{aligned} \tag{2.31}$$

Now, the derivatives are set up to be solved in the computational domain, therefore, the solver can benefit from employing simple finite differences through equal spacing. The approach is to first set up the grid points at the boundary in the physical domain, and then solve for the solution in the computational domain. The forcing functions are only added when there is a need to include clustering around points of grid lines as shown in Figure 2.13. The $P(\xi,\eta)$ is the control function for the ξ direction and $Q(\xi,\eta)$ is the control function for the η direction. These equations can be solved using a Gauss-Seidel approach.



Attraction to line ξ_i due to $P(\xi,\eta)$

Attraction to point (ξ_j,η_j) due to $P(\xi,\eta)$

Attraction to line η_i due to $Q(\xi,\eta)$

Attraction to point (ξ_j,η_j) due to $Q(\xi,\eta)$

Figure 2.13: Clustering effect by introducing $P(\xi,\eta)$ and $Q(\xi,\eta)$ [61]

2.4.5 Hyperbolic Equation Approach

Hyperbolic equation is newer than elliptical and parabolic equation approaches. The main benefit of this method is that it is much faster in mesh generation time because it uses a front marching method (i.e., traversing the solution from the inner boundary up to the outer boundary). Also, it is very applicable to CFD because the grid generated using this method is inherently orthogonal at the boundaries which is very useful for resolving the boundary layer in a viscous flow. The disadvantage of this method is that the scheme itself is not very stable. To adjust for instabilities, it is possible to introduce artificial dissipation which will be shown later to post-process the mesh at the output step. For a general review of the hyperbolic equation approach, the review paper by Sørensen is a good source [62].

The fundamental concept that builds up the hyperbolic system is two constraints. The first one is the orthogonality at the boundaries. This is achieved through the following equation,

$$\vec{r}_\xi \cdot \vec{r}_\eta = 0 \quad \rightarrow \quad x_\xi \cdot x_\eta + y_\xi \cdot y_\eta = 0 \tag{2.32}$$

This is the definition of orthogonality from linear algebra where the dot product between two vectors is zero when they are perpendicular to each other. The second constraint is on the Jacobian determinant as shown in section 2.2.1, and it is given as follows,

$$J = det[J_2] = x_\xi y_\eta - x_\eta y_\xi = F(\xi, \eta) \tag{2.33}$$

Here, the cell area function is denoted as $F(\xi, \eta)$ and is equal to the Jacobian determinant. The cell area function is defined as follows,

$$F_{i,j} = \triangle c_{i,j} \cdot \triangle s_{i,j} \tag{2.34}$$

$$\triangle c_{i,j} = 0.5 * \left(\left| \vec{r}_{i,j} - \vec{r}_{i-1,j} \right| + \left| \vec{r}_{i+1,j} - \vec{r}_{i,j} \right| \right) \tag{2.35}$$

$$\triangle s_{i,j} = \triangle s_1 * (1 + \epsilon)^{j-1}, \tag{2.36}$$

where s_1 is the spacing of the second grid from the first boundary grid along $\eta = 1$ and ϵ is the stretching ratio along the η direction as specified by the user.

2.4.6 Solution to the Hyperbolic Equation Approach

The solution x and y are represented by a known state, x^k and y^k, as well as an increment, $\triangle x$ and $\triangle y$, denoted as follows,

$$x = x^k + \triangle x \tag{2.37}$$

$$y = y^k + \triangle y \tag{2.38}$$

Then, equation 2.37 and 2.38 are substituted into the two constraints (equation 2.32 and 2.33) which describe the hyperbolic system as follows,

$$(x^k + \triangle x)_\xi (x^k + \triangle x)_\eta + (y^k + \triangle y)_\xi (y^k + \triangle y)_\eta = 0 \tag{2.39}$$

$$(x^k + \triangle x)_\xi (y^k + \triangle y)_\eta - (x^k + \triangle x)_\eta (y^k + \triangle y)_\xi = F(\xi, \eta) + F^k(\xi, \eta) \tag{2.40}$$

Doing some simplification, the following is obtained,

$$x_\eta x_\xi^k + x_\eta^k x_\xi + y_\xi y_\eta^k + y_\xi^k y_\eta = 0 \tag{2.41}$$

$$x_\xi y_\eta^k + y_\eta x_\xi^k - x_\eta y_\xi^k - y_\xi x_\eta^k = F(\xi, \eta) + F^k(\xi, \eta) \tag{2.42}$$

Then, the previous equations can be grouped into the following system of equations,

$$\begin{bmatrix} x_\eta^k & y_\eta^k \\ y_\eta^k & -x_\eta^k \end{bmatrix} \begin{bmatrix} x_\xi \\ y_\xi \end{bmatrix} + \begin{bmatrix} x_\xi^k & y_\xi^k \\ -y_\xi^k & x_\xi^k \end{bmatrix} \begin{bmatrix} x_\eta \\ y_\eta \end{bmatrix} = \begin{bmatrix} 0 \\ F(\xi, \eta) + F^k(\xi, \eta) \end{bmatrix} \tag{2.43}$$

In addition, equation 2.43 can be further simplified by introducing the following terms while also using a first-order backward finite difference for η and second-order central difference for ξ derivatives which yields,

$$R = \begin{bmatrix} x \\ \\ y \end{bmatrix} \quad A = \begin{bmatrix} x_\eta^k & y_\eta^k \\ \\ y_\eta^k & -x_\eta^k \end{bmatrix} \quad B = \begin{bmatrix} x_\xi^k & y_\xi^k \\ \\ -y_\xi^k & x_\xi^k \end{bmatrix} \quad H = \begin{bmatrix} 0 \\ \\ F(\xi,\eta) + F^k(\xi,\eta) \end{bmatrix} \tag{2.44}$$

Then, the following is obtained,

$$[A]\frac{R_{i+1,j} - R_{i-1,j}}{2\triangle\xi} + [B]\frac{R_{i,j} - R_{i,j-1}}{\triangle\eta} = H_{i,j} \tag{2.45}$$

Multiplying on both sides by the inverse matrix of ”B” yields the following,

$$[B]^{-1}[A]\frac{R_{i+1,j} - R_{i-1,j}}{2\triangle\xi} + [I]\frac{R_{i,j} - R_{i,j-1}}{\triangle\eta} = [B]^{-1}H_{i,j} \tag{2.46}$$

At this stage, all variables with the superscript *k* are known states. Since an outward marching scheme from the first boundary at $\eta = 1$ is employed, it is apparent that the matrices *A* and *B* are evaluated at the ($\eta - 1^{\text{th}}$) grid lines. Further simplification of equation 2.46 yields,

$$-\frac{1}{2\triangle\xi}\left([B]^{-1}[A]\right)_{i,j-1} R_{i-1,j} + \frac{R_{i,j}}{\triangle\eta} + \frac{1}{2\triangle\eta}\left([B]^{-1}[A]\right)_{i,j-1} R_{i+1,j} = [B]^{-1}_{i,j-1}H_{i,j} + \frac{R_{i,j-1}}{\triangle\eta} \tag{2.47}$$

Notice here that it is possible to regroup certain terms to simplify the previous equation in the following way,

$$[AA] = -\frac{1}{2\triangle\xi}\left([B]^{-1}[A]\right)_{i,j-1} \tag{2.48}$$

$$[BB] = [I] \tag{2.49}$$

$$[CC] = -[AA] \tag{2.50}$$

$$[DD] = [B]^{-1}_{i,j-1}[H]_{i,j} + R_{i,j-1} \tag{2.51}$$

Finally, the following formulation is obtained,

$$[AA]R_{i-1,j} + [BB]R_{i,j} + [CC]R_{i+1,j} = [DD]_{i,j}. \tag{2.52}$$

And, from this formulation, a block tridiagonal matrix appears. Thomas algorithm could have been employed to solve equation 2.52 if it was a simple tridiagonal matrix, however, since this is a block tridiagonal matrix, the block tridiagonal matrix version of the Thomas algorithm would be required and the matrix is set up as follows,

$$\begin{bmatrix} [BB]_2 & [CC]_2 & & & & \\ [AA]_3 & [BB]_3 & [CC]_3 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & [AA]_{NX-2} & [BB]_{NX-2} & [CC]_{NX-2} \\ & & & & [AA]_{NX-1} & [BB]_{NX-1} \end{bmatrix} \begin{bmatrix} R_2 \\ R_3 \\ \vdots \\ \vdots \\ R_{NX-2} \\ R_{NX-1} \end{bmatrix} = \begin{bmatrix} [\mathrm{DD}]'_2 \\ [\mathrm{DD}]_3 \\ \vdots \\ \vdots \\ [\mathrm{DD}]_{NX-2} \\ [\mathrm{DD}]'_{NX-1} \end{bmatrix}$$

For the first and last entry in the $[DD]$ column matrix, additional terms are required because this is a block tridiagonal system. At the edges of the grid line, boundary grid points are already specified before solving for the solution, hence, these are known ”grid points” on the other side (in the $[DD]$ matrix) which can be added to it. Hence,

$$[DD]'_2 = [DD]_2 - [AA]_2 R_1 \tag{2.53}$$

$$[DD]'_{NX-1} = [DD]_{NX-1} - [CC]_{NX-1} R_{NX} \tag{2.54}$$

Lastly, the first-order derivatives in the η direction are already known and should be specified. However, specifying them is challenging since it requires the grid point at the next η level to complete the first-order finite difference, hence, what is generally done is to employ the two constraints that make up the hyperbolic system and solve them for the first-order derivatives along the "η" direction. This yields,

$$x_\eta^k = -\frac{y_\xi^k F^k}{(x_\xi^k)^2 + (y_\xi^k)^2} \tag{2.55}$$

$$y_\eta^k = \frac{x_\xi^k F^k}{(x_\xi^k)^2 + (y_\xi^k)^2} \tag{2.56}$$

This sums up the discussion for the solution. In the next section, the artificial dissipation term that is added to increase stability in the scheme will be discussed.

2.4.7 Artificial Dissipation

When solving the block tridiagonal matrix, instability can occur due to the formulation of the hyperbolic PDE, hence, an artificial dissipation term can be added next to the $[DD]$ column matrix in equation 2.53. This yields,

$$[AA]R_{i-1,j} + [BB]R_{i,j} + [CC]R_{i+1,j} = [DD]_{i,j} + \phi \tag{2.57}$$

where ϕ is given by the following,

$$\phi = \beta(R_{i+2,j} - 4R_{i+1,j} + 6R_{i,j} - 4R_{i-1,j} + R_{i-2,j}) \tag{2.58}$$

Note that β is the artificial dissipation factor which is a user-defined parameter. One issue here is that the ϕ term requires the points that are two grid spaces on the left and right, which means that the algorithm will have to start sweeping the solution at the third point or $\xi = 3$ to make sure that the artificial dissipation term can be computed correctly. Multiple ways can be used to compute this, one standard approach is to use a larger grid which contains two extra grid layers outside the standard boundary for computing the artificial dissipation term. This leads to a larger grid size compared to the original grid. Another possible way is to generate ghost points on the outside of the boundaries which can be computed on the fly as the algorithm moves across the boundary grid points. These ghost points would be generated using the first two layers generated using transfinite interpolation and then reflected outside the boundaries using simple geometrical techniques. With this technique, the original grid can be conserved. More information about this can be found in [63].

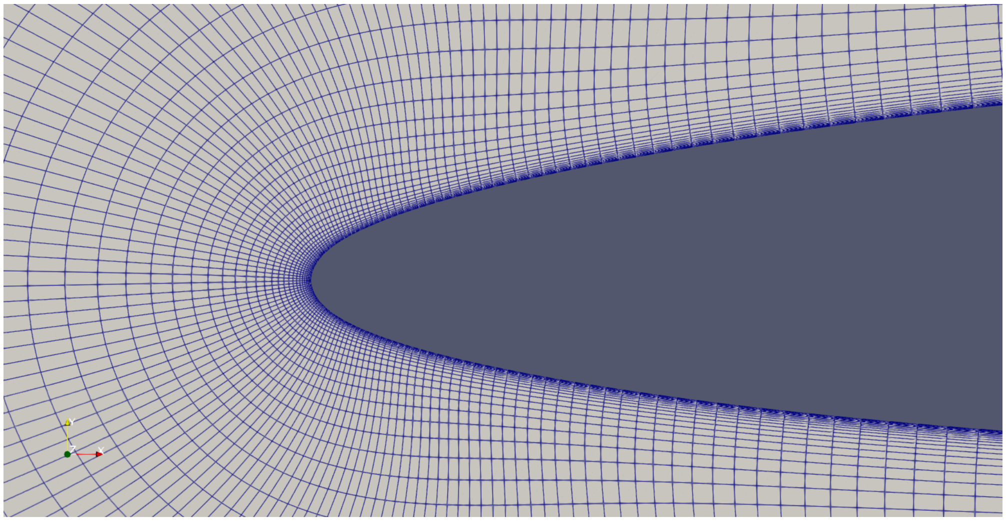


Figure 2.14: Example of Hyperbolic Equation Grid Generation at the leading edge of airfoil

Here are examples of the various meshing techniques for structured quadrilateral mesh.

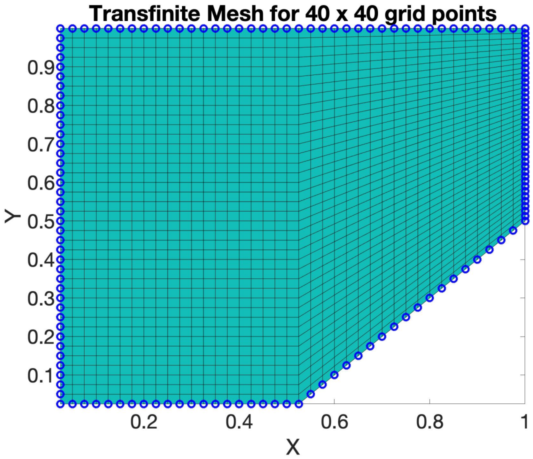


Figure 2.15: Transfinite Interpolation

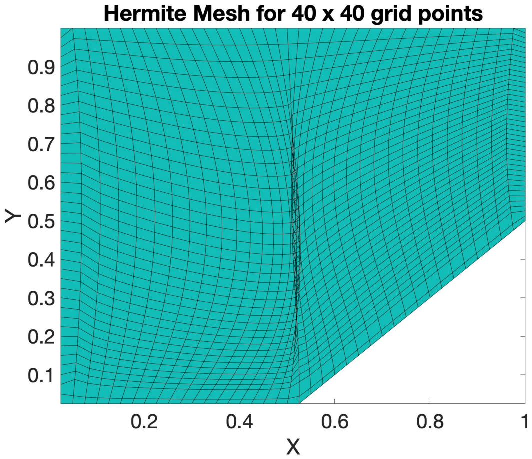


Figure 2.16: Hermite Interpolation

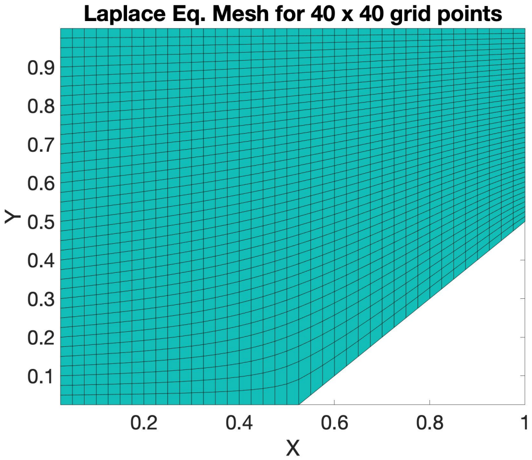


Figure 2.17: Laplace Equation Solution

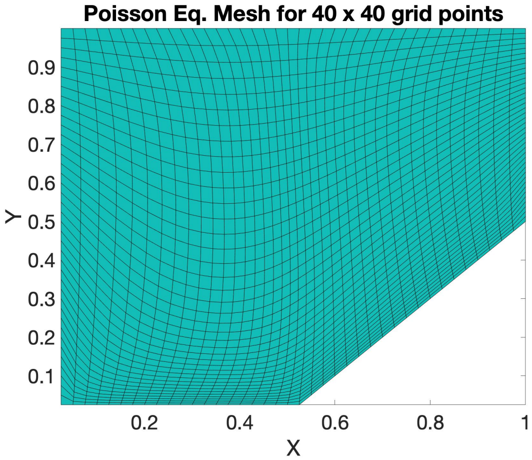


Figure 2.18: Poisson Equation Solution

2.5 Metric based Inner Product, Norm, and Distance Definition

A general introduction to quadrilateral mesh generation has been established, now, the following section will focus on metric tensors and how they are applied to anisotropic mesh adaptation.

First, a metric or a metric tensor $\mathcal{M}$ is a symmetric definite positive matrix. When applied to a dot product between two vectors, **u** and **v**, the definition is then written as,

$$\langle \boldsymbol{u} \, , \boldsymbol{v} \rangle_{\mathcal{M}} = \langle \boldsymbol{u} \, , \mathcal{M}\boldsymbol{v} \rangle = \boldsymbol{u}^T \, \mathcal{M}\boldsymbol{v} \tag{2.59}$$

Now, when applying the metric to a norm on a vector **u**, the norm is defined as,

$$\|\boldsymbol{u}\|_{\mathcal{M}} = \sqrt{\langle \boldsymbol{u} \, , \boldsymbol{u} \rangle_{\mathcal{M}}} \tag{2.60}$$

The simplest form or example of a metric tensor is an identity matrix, such as,

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.61}$$

This as a result simply recovers the Euclidean space $\mathbb{R}^2$ where the dot product as well as the norm in equation 2.59 and 2.60 is recovered as,

$$\langle \boldsymbol{u} \, , \boldsymbol{v} \rangle_{I_2} = \langle \boldsymbol{u} \, , I_2\boldsymbol{v} \rangle = \boldsymbol{u}^T \, I_2\boldsymbol{v} = \boldsymbol{u}^T \, \boldsymbol{v} \tag{2.62}$$

$$\|\boldsymbol{u}\|_{I_2} = \sqrt{\langle \boldsymbol{u} \, , \boldsymbol{u} \rangle_{I_2}} = \sqrt{\boldsymbol{u}^T \, \boldsymbol{u}} \tag{2.63}$$

Note that, in the metric space, it is also possible to define a distance formula between two vectors which can be written as,

$$d_{\mathcal{M}}(\boldsymbol{u}, \boldsymbol{v}) = \|\boldsymbol{u} - \boldsymbol{v}\|_{\mathcal{M}} \tag{2.64}$$

Similarly, the Euclidean distance formula can be recovered by using an identity matrix for the metric tensor $\mathcal{M}$.

2.5.1 Euclidean Metric Space

A Euclidean metric space is a metric space spanned by a constant metric tensor on its domain Ω. In particular, this means that the metric tensor $\mathcal{M}$ in equation 2.59, 2.60 and 2.64 is constant for any given vectors **u** and **v**. In other words, the metric tensor $\mathcal{M}$ does not depend on the coordinate system. Namely, it is defined as,

$$\mathcal{M} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \tag{2.65}$$

With this formulation, the distance formula can still be evaluated using equation 2.64 which at the same time, acts as the length computed between two vectors. We will refer to this as the length of the edge evaluated under the Euclidean metric space, which is defined as,

$$l_{\mathcal{M}}(\boldsymbol{ab}) = d_{\mathcal{M}}(\boldsymbol{a}, \boldsymbol{b}) = \|\boldsymbol{a} - \boldsymbol{b}\|_{\mathcal{M}} \tag{2.66}$$

2.5.2 Riemmanian Metric Space

A Riemannian metric space, as opposed to a Euclidean metric space, does not have a constant metric tensor spanning the whole domain Ω. In contrast, a metric *field* is instead introduced and it varies smoothly in the whole domain Ω. As a result, the metric tensor

$\mathcal{M}$ can then written as,

$$\mathcal{M} = \begin{bmatrix} m_1(x,y) & m_2(x,y) \\ m_3(x,y) & m_4(x,y) \end{bmatrix} \tag{2.67}$$

The continuous mesh framework discussed in Chapter 4 will be based on the Riemannian metric space. Now, since the metric tensor $\mathcal{M}$ is varying within the whole domain Ω, the length of an edge is no longer equal to the distance due to the variation of the metric tensor between two vectors. Instead, the length of an edge is defined using an integral for $t \in [0,1]$,

$$l_{\mathcal{M}}(\boldsymbol{ab}) = \int_0^1 \|\gamma'(t)\|_{\mathcal{M}}\, dt = \int_0^1 \sqrt{\boldsymbol{ab}^T\, \mathcal{M}(\boldsymbol{a} + t\, \boldsymbol{ab})\; \boldsymbol{ab}}\, dt \tag{2.68}$$

To visualize a metric field, it is very common to use the metric norm as an indicator for demonstrating how the metric tensor at a specific location in the domain varies across the space. The easiest way to do this is to build a unit circle around the chosen location using the metric norm. To demonstrate this concept, recall equation 2.60 and equalize it to 1, then squaring both sides yields,

$$\|\boldsymbol{u}\|_{\mathcal{M}} = \langle \boldsymbol{u}\,, \boldsymbol{u}\rangle_{\mathcal{M}} = 1 \tag{2.69}$$

Now, expanding it results in,

$$\boldsymbol{u}^T\, \mathcal{M}\boldsymbol{u} = (\boldsymbol{x} - \boldsymbol{a})^T\, \mathcal{M}(\boldsymbol{x} - \boldsymbol{a}) = 1, \tag{2.70}$$

where **x** is an arbitrary position in the domain (i.e. the end of vector **u**), and **a** is the center of the unit circle (i.e. starting point of vector **u**). An interesting note here is that the metric tensor $\mathcal{M}$ can be further simplified through eigenvalue decomposition, which results in,

$$\boldsymbol{u}^T\, (\mathcal{R}\Lambda\mathcal{R}^T)\, \boldsymbol{u} = (\boldsymbol{x} - \boldsymbol{a})^T \mathcal{R}\,\Lambda\,\mathcal{R}^T(\boldsymbol{x} - \boldsymbol{a}) = 1 \tag{2.71}$$

$$(\boldsymbol{x}-\boldsymbol{a})^T \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}^T (\boldsymbol{x}-\boldsymbol{a}) = 1, \tag{2.72}$$

where $\mathcal{R}$ is an orthonormal matrix and Λ is a diagonal matrix with its entries being the eigenvalues of the metric tensor $\mathcal{M}$. Note as well that the eigenvalue entries are strictly positive. Consequently, equation 2.72 becomes,

$$(\tilde{\boldsymbol{x}}-\tilde{\boldsymbol{a}})^T \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} (\tilde{\boldsymbol{x}}-\tilde{\boldsymbol{a}}) = 1 \tag{2.73}$$

which is equal to,

$$\sum_{i=1}^{2} \left(\frac{\tilde{x}_i - \tilde{a}_i}{h_i} \right)^2 = 1 \tag{2.74}$$

Equation 2.74 reassembles very much the equation of an ellipse with the length of its axes being described by h_i which is equal to $\lambda_i^{-1/2}$. Lastly, there exists a natural mapping between the metric norm and the L_2 norm which is depicted through the eigenvalue matrix. The similarity between both norms comes down to the mapping between the metric tensor $\mathcal{M}$ and the identity matrix I_2. This relationship is written as follows,

$$\|\boldsymbol{u}\|_{\mathcal{M}} = \|\mathcal{M}^{-\frac{1}{2}}\boldsymbol{u}\|_2 \tag{2.75}$$

Figure 2.19 shows an example of a smoothly varying metric tensor applied to an Euclidean space where the unit ball is drawn at the chosen red points. All in all, the purpose of the Riemannian metric space is to compute the edge length using equation 2.68. For more information about Riemannian metric space, please refer [7, 18, 39].

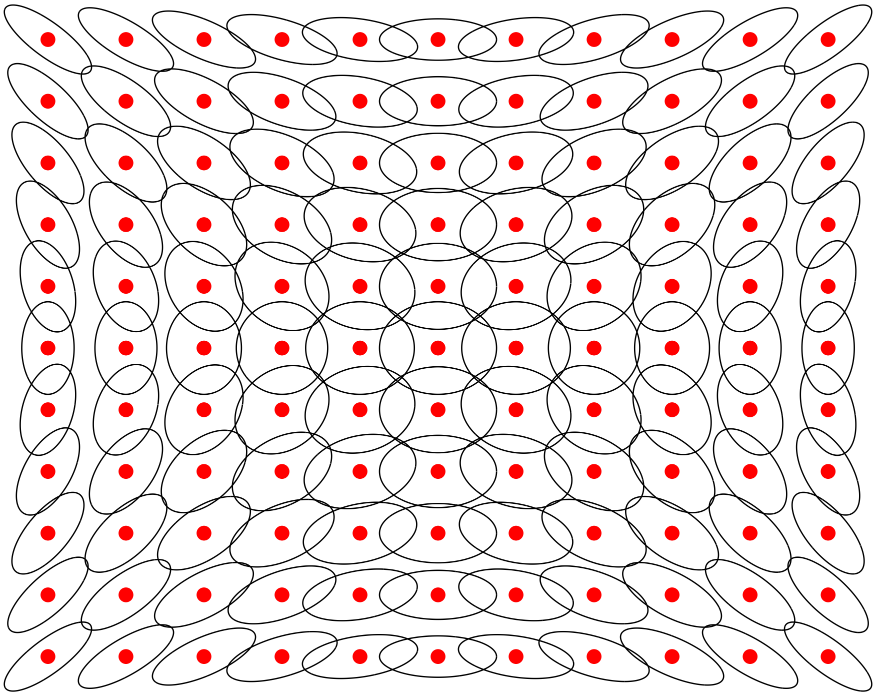


Figure 2.19: Unit ellipse at each position depicted by red points

2.6 Frame Fields

Frame fields are similar to Riemmanian metric fields as they share many common concepts that can be applied in both cases. This concept was originally introduced by Panozzo et al. in their work [27, 28]. The beauty of this concept is that the frame fields inherently contain information about the target orientation as well as the target size of a quadrilateral through defining a frame at every location on a smooth surface $\mathcal{S}$ in $\mathbb{R}^2$. This is done through a pair of linearly independent vectors that are ordered in a cyclic counterclockwise fashion, namely $\mathbf{v}$ and $\mathbf{w}$, which defines each frame at on a tangent plane $\mathcal{T}_p\mathcal{S}$ at a position $\mathbf{p}$ on $\mathcal{S}$ through,

$$f_p =< \boldsymbol{v}, \boldsymbol{w}, -\boldsymbol{v}, -\boldsymbol{w} > = \ V < \boldsymbol{e_1}, \boldsymbol{e_2}, -\boldsymbol{e_1}, -\boldsymbol{e_2} >, \tag{2.76}$$

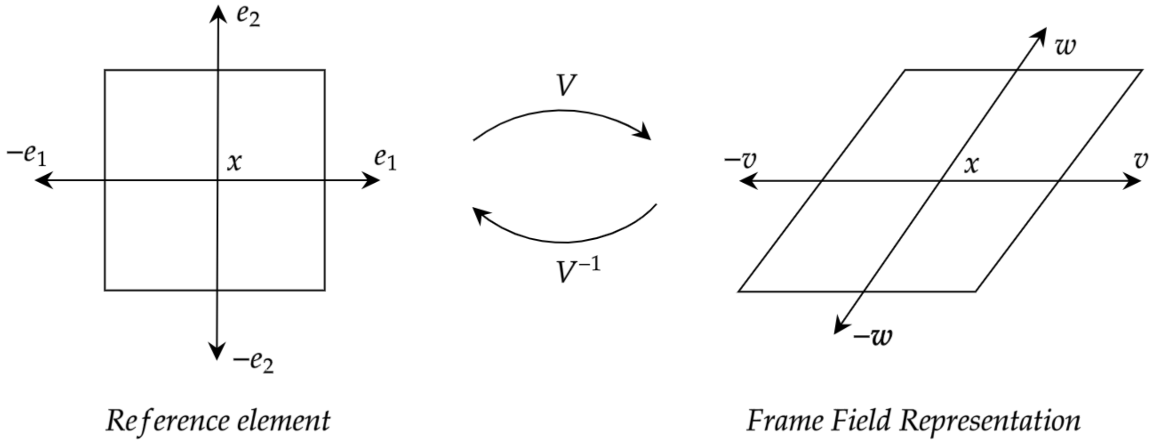


Figure 2.20: Frame field Mapping

where V is a linear mapping between the reference element and the physical element. Also, the matrix V is defined as follows through polar decomposition,

$$V = \begin{bmatrix} \boldsymbol{v} & \boldsymbol{w} \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta_1) & \cos(\theta_2) \\ \sin(\theta_1) & \sin(\theta_2) \end{bmatrix}}_{R(\theta)} \begin{bmatrix} h_1 & 0 \\ 0 & h_2 \end{bmatrix} \tag{2.77}$$

In this formulation, the parameters h_1 and h_2 are the lengths of the respective vectors **v** and **w**. Also, θ_1 and θ_2 are the angles associated with vectors **v** and **w** where the first matrix containing angles is known as the rotation matrix $R(\theta)$. In this work, right-angled quadrilaterals are of interest for the mesh adaptation, and therefore, assumptions are made to generate frames that contain vectors **v** and **w** that are orthonormal. To achieve this, the angle θ_2 is simply shifted by 90° over θ_1 resulting in $\theta_2 = \theta_1 + 90°$.

Now, by introducing an average size $h = \sqrt{h_1 h_2} = \sqrt{\det(V)}$, and the anisotropy term $\rho = \frac{h_1}{h_2}$, equation 2.77 can be further reduced to for the forward (equation 2.78) and backward (equation 2.79) mapping,

$$V = h\,R(\theta)\begin{bmatrix}\sqrt{\rho} & 0 \\ 0 & \frac{1}{\sqrt{\rho}}\end{bmatrix} \tag{2.78}$$

$$V^{-1} = \frac{1}{h}\begin{bmatrix}\frac{1}{\sqrt{\rho}} & 0 \\ 0 & \sqrt{\rho}\end{bmatrix} R(-\theta) \tag{2.79}$$

In a similar fashion to equation 2.75, the linear mapping matrix V can be related to the L_∞ norm (L_2 norm here) through the following relationship,

$$\|\boldsymbol{u}\|_f = \|V^{-1}\boldsymbol{u}\|_\infty \tag{2.80}$$

However, in practice, L_∞ norm is non-differentiable, and therefore, a selection of the *p* value must be made to approximate the L_∞ norm. In [39], Maclean and Nadajarah found that a *p* value of six is sufficient since values beyond six did not yield further benefits. Hence in this work, *p* is selected as six. Additionally, using this formulation of L_p norm, it follows very well the definition of the L_p-CVT framework introduced by Lévy and Liu in [48], which will be utilized with frame fields for the mesh adaptation framework discussed in this work. Figure 2.21 shows the various norms using different values of *p* where an increase in the p value of the L_p-norm gradually changes the shape of the circle into a rectangle which helps in generating rectangular triangles to eventually form quadrilaterals in the output mesh.

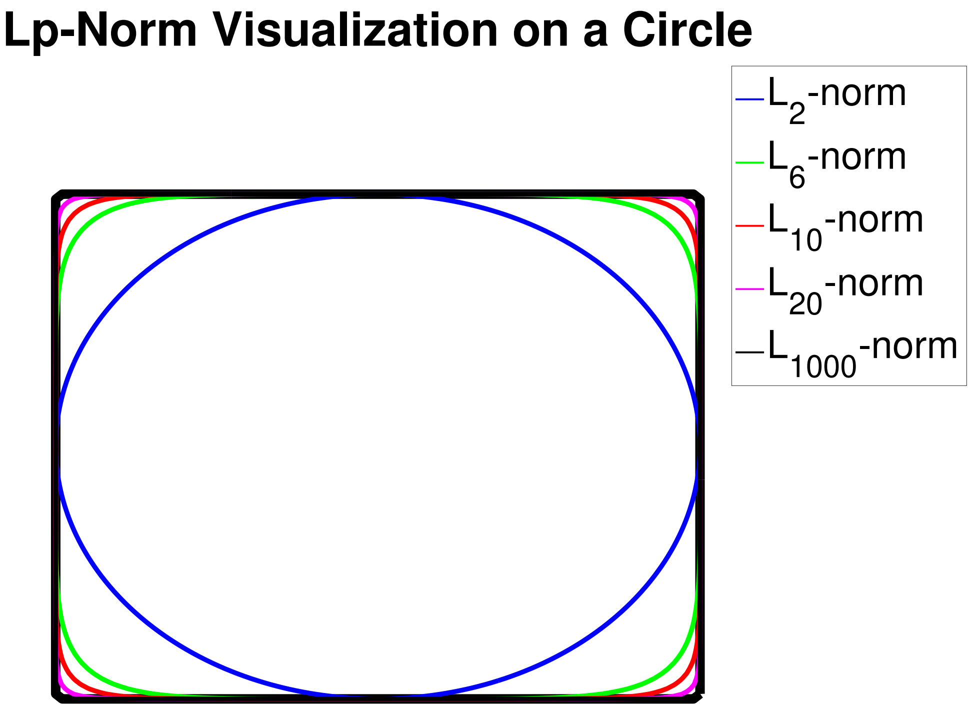


Figure 2.21: Comparison of various L_p-norm on a Circle

2.7 Metric-based Mesh Adaptation

From the previous discussion, it is established that both a metric tensor as well as a frame provide information on anisotropy when related to a norm. The idea behind metric-based mesh adaptation is to generate an isotropic unit mesh with edges of unit lengths in the metric space which results in an anisotropic mesh in the physical space [12]. This method is utilized in various software such as BAMG [64] and Yams [64]. This can be formulated as follows from [17],

$$\mathcal{T}_h = \operatorname{argmin} \sum_{\boldsymbol{e} \in \mathcal{T}_h'} \left(\|\boldsymbol{e}\|_{\mathcal{M}} - C \right)^2 , \tag{2.81}$$

where the edge length is computed using equation 2.60 and the parameter C is a scaling constant that is user-defined. As demonstrated in [17], Dolejší uses a scaling constant of $\sqrt{3}$ which represents the edge length of an equilateral triangle within a unit circle. Figure 2.23 shows an example of a triangle generated based on a metric tensor,

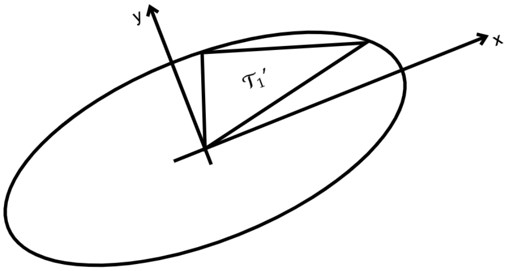


Figure 2.22: Metric tensor ellipse with a single corresponding generated triangle

Another interesting fact is that the triangle generated in Figure 2.23 is not unique [17, 18, 19, 65]. A visual inspection of the metric tensor ellipse shows that the following triangle configuration (i.e.$\mathcal{T}_1', \mathcal{T}_2', \mathcal{T}_3', \mathcal{T}_4'$) is also possible,

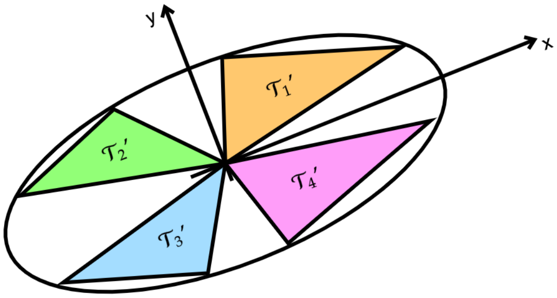


Figure 2.23: Metric tensor ellipse with possible configuration of generated triangles

Therefore, depending on the metric tensor which adds anisotropy to the mesh and the method of choice for metric-based adaptation, the output mesh will very likely not be unique for a given initial mesh. For this work, the L_p-CVT technique will be used to solve a similar problem in the L_p norm with concepts from Voronoi Diagram.

2.8 L_p-CVT Mesh Generator

In this work, L_p-CVT is used as the mesh adaptation framework which stands for Centroidal Voronoi Tesselation in the L_p norm. As discussed previously in section 2.3, L_p-CVT follows very similarly the principle of CVT, however, instead of using the L_2 norm for defining the distance between the contour of the cell and its centroid, this method aims to build Voronoi cells through an arbitrary p norm while taking into account anisotropy from a background metric field $M(\boldsymbol{y})$.

2.8.1 Local Facet Energy Computation

Before jumping into the theory behind L_p-CVT, it is important to first understand the structure of a Voronoi cell. A Voronoi cell is made of facets which are triangular structures within a cell called facet. Facets are the principal component (i.e. f_1 and f_4 in Figure 2.24) for computing energy and gradient on Voronoi cells when solving for the L_p-CVT.

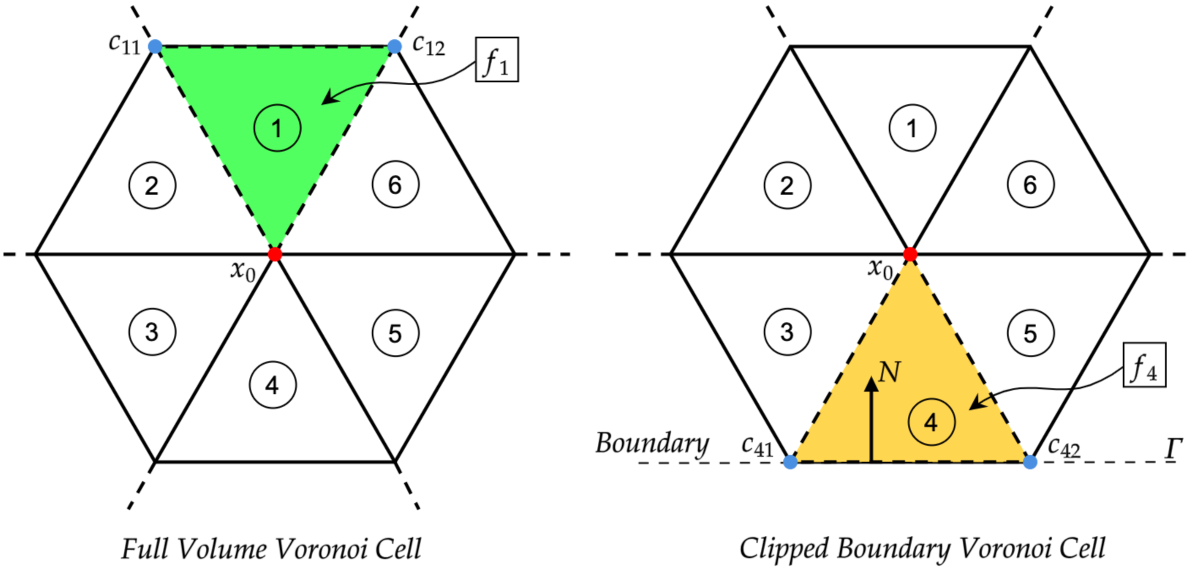


Figure 2.24: Voronoi Cell with 6 facets - Left: Full Volume Voronoi Cell, Right: Clipped Boundary Voronoi Cell

First, the local energy of a facet inside a Voronoi Cell is given by the following,

$$\begin{aligned} I_k &= \int_{fk} \|M(\boldsymbol{y})(\boldsymbol{y} - \boldsymbol{x_i})\|_p^p d\boldsymbol{y} \\ &= \int_{f'} \|M(\boldsymbol{T(\xi)})(\boldsymbol{T(\xi)} - \boldsymbol{x_i})\|_p^p J \, d\boldsymbol{\xi} \\ &= \int_{f'} \|\boldsymbol{F(y)}\|_p^p J \, d\boldsymbol{\xi}, \end{aligned} \tag{2.82}$$

where $M(\boldsymbol{y})$ is the background metric term evaluated at a location $\boldsymbol{y}$ which defines the target anisotropy, and $\boldsymbol{x_i}$ is the node in which the Voronoi Cell is built around (i.e. x_0 in Figure 2.24). Now, $\boldsymbol{y}$ are arbitrary points inside a facet. For the evaluation of the integral by Gaussian Quadrature, $\boldsymbol{y}$ are selected to be the quadrature points in the facets. Note that the term $\boldsymbol{T(\xi)}$ is the mapping between the reference triangle to the actual facet in the physical domain and therefore, quadrature points on the reference triangle do not change which facilitates the computation of the integral. Finally, J is the determinant of the Jacobian defining the linear transformation between the reference and physical domain which maps the reference triangle to the actual triangle and it is identical to equation 2.2. Lastly, $\boldsymbol{F(y)}$ is equal to $M(\boldsymbol{T(\xi)})(\boldsymbol{T(\xi)} - \boldsymbol{x_i})$ which is written as a shorthanded term for later derivations. These terms will be derived in the following section.

2.8.2 Total Energy and Derivative Computation

The total energy in the Voronoi diagram, E_{L_p} is therefore the sum of the energy generated by each Voronoi cell, Ω_i, which is in itself equal to the sum of all facets, f_k, in the cell, which is shown as follows,

$$E_{L_p} = \sum_i \sum_{f_k \in \Omega_i} I_k = \sum_i \sum_{f_k \in \Omega_i} \int_{f'} \|\boldsymbol{F(y)}\|_p^p J \, d\boldsymbol{\xi} \tag{2.83}$$

Similarly, the total energy gradient is given by,

$$\frac{\partial E}{\partial \boldsymbol{x_i}} = \sum_{f_k \in \Omega_i} \frac{\partial I_k}{\partial \boldsymbol{x_i}} + \sum_{\boldsymbol{c_j} \in \Omega_i} \sum_{f_k \ni \boldsymbol{c_i}} \frac{\partial I_k}{\partial \boldsymbol{c_j}} \frac{\partial \boldsymbol{c_j}}{\partial \boldsymbol{x_i}} \tag{2.84}$$

The gradient can then be separated into three terms. The first term on the left, $\frac{\partial I_k}{\partial x_i}$, is the sum of the derivative of the local energy of a facet with respect to its Voronoi site. Then, the second term on the right, $\frac{\partial I_k}{\partial \boldsymbol{c_j}}$, is the partial derivative of the local energy of a facet with respect to a Voronoi vertex (i.e. c_{11} and c_{12} in Figure 2.24). Finally, the third term, $\frac{d\boldsymbol{c_j}}{d\boldsymbol{x_i}}$, is the chain rule for the derivative of the vertex position with respect to the Voronoi site. The terms are defined as follows,

$$\frac{\partial I_k}{\partial \boldsymbol{x_i}} = \int_{f'} = \frac{\|\boldsymbol{F}\|_p^p}{\partial \boldsymbol{F}} \left[\frac{\partial M}{\partial \boldsymbol{T}} \frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x_i}} (\boldsymbol{T}(\boldsymbol{\xi}) - \boldsymbol{x_i}) + M(\boldsymbol{T}(\boldsymbol{\xi})) \left(\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x_i}} - 1 \right) \right] J + \|\boldsymbol{F}\|_p^p \frac{\partial \boldsymbol{J}}{\partial \boldsymbol{x_i}} d\boldsymbol{\xi} \tag{2.85}$$

$$\frac{\partial I_k}{\partial \boldsymbol{c_j}} = \int_{f'} = \frac{\|\boldsymbol{F}\|_p^p}{\partial \boldsymbol{F}} \left[\frac{\partial M}{\partial \boldsymbol{T}} \frac{\partial \boldsymbol{T}}{\partial \boldsymbol{c_j}} (\boldsymbol{T}(\boldsymbol{\xi}) - \boldsymbol{x_i}) + M(\boldsymbol{T}(\boldsymbol{\xi})) \left(\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{c_j}} \right) \right] J + \|\boldsymbol{F}\|_p^p \frac{\partial \boldsymbol{J}}{\partial \boldsymbol{c_j}} d\boldsymbol{\xi} \tag{2.86}$$

Now, the derivative of the vertex position with respect to the Voronoi site, $\frac{d\boldsymbol{c_j}}{d\boldsymbol{x_i}}$ in two-dimensional space is derived based on the derivation of Lévy and Liu for the case of three-dimensional space [48].

Let the circumcenter, C_o, of a triangle be defined by the bisectors on each of the three sides of the triangle as defined in Figure 2.25. By inspection, two bisectors are sufficient for defining the circumcenter since in a triangle, the circumcenter is implicitly defined by the intersection of the first two bisectors. The third bisector can be drawn by connecting the circumcenter to the third side of the triangle. Now, let x_0 be the origin of the triangle or a Voronoi site. Then, we can define the following,

$$\overrightarrow{AB} = \boldsymbol{x_1} - \boldsymbol{x_0} \tag{2.87}$$

$$\overrightarrow{AC} = \boldsymbol{x_2} - \boldsymbol{x_0} \tag{2.88}$$

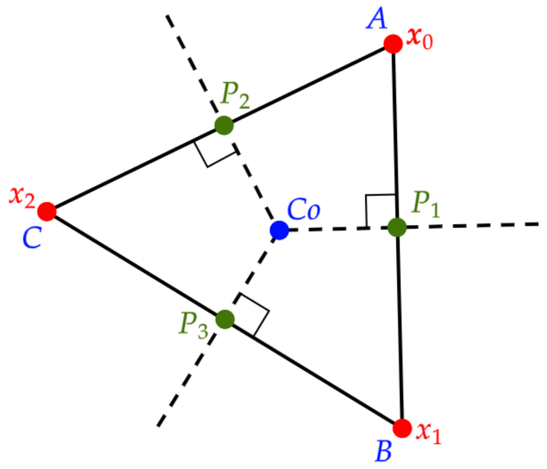


Figure 2.25: Triangle with 3 Bisectors

Also, the following can be established with the definition of the bisectors,

$$\overrightarrow{AB} \cdot (\boldsymbol{C_o} - \boldsymbol{P_1}) = \overrightarrow{0} \tag{2.89}$$

$$\overrightarrow{AC} \cdot (\boldsymbol{C_o} - \boldsymbol{P_2}) = \overrightarrow{0} \tag{2.90}$$

Distributing the dot product in equation 2.89 and 2.90, and manipulating the equations gives the following,

$$\overrightarrow{AB} \cdot \boldsymbol{C_o} = \overrightarrow{AB} \cdot \boldsymbol{P_1} \tag{2.91}$$

$$\overrightarrow{AC} \cdot \boldsymbol{C_o} = \overrightarrow{AC} \cdot \boldsymbol{P_2} \tag{2.92}$$

Now, rewriting equation 2.91 and 2.92 as a system of equations, and replacing the corresponding variables yields the following,

$$\begin{bmatrix} [\boldsymbol{x_1} - \boldsymbol{x_0}]^T \\ [\boldsymbol{x_2} - \boldsymbol{x_0}]^T \end{bmatrix} \boldsymbol{C_o} = \begin{bmatrix} [\boldsymbol{x_1} - \boldsymbol{x_0}] \cdot [\boldsymbol{x_1} + \boldsymbol{x_0}]/2 \\ [\boldsymbol{x_2} - \boldsymbol{x_0}] \cdot [\boldsymbol{x_2} + \boldsymbol{x_0}]/2 \end{bmatrix} \tag{2.93}$$

Now, let $\boldsymbol{A}$ and $\boldsymbol{B}$ denote the following,

$$\boldsymbol{A} = \begin{bmatrix} [\boldsymbol{x_1} - \boldsymbol{x_0}]^T \\ [\boldsymbol{x_2} - \boldsymbol{x_0}]^T \end{bmatrix} \tag{2.94}$$

$$\boldsymbol{B} = \begin{bmatrix} [\boldsymbol{x_1} - \boldsymbol{x_0}] \cdot [\boldsymbol{x_1} + \boldsymbol{x_0}]/2 \\ [\boldsymbol{x_2} - \boldsymbol{x_0}] \cdot [\boldsymbol{x_2} + \boldsymbol{x_0}]/2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} [\boldsymbol{x_1}^2 - \boldsymbol{x_0}^2] \\ [\boldsymbol{x_2}^2 - \boldsymbol{x_0}^2] \end{bmatrix} \tag{2.95}$$

Then,

$$\boldsymbol{Co} = \begin{bmatrix} [\boldsymbol{x_1} - \boldsymbol{x_0}]^T \\ [\boldsymbol{x_2} - \boldsymbol{x_0}]^T \end{bmatrix}^{-1} \begin{bmatrix} [\boldsymbol{x_1} - \boldsymbol{x_0}] \cdot [\boldsymbol{x_1} + \boldsymbol{x_0}]/2 \\ [\boldsymbol{x_2} - \boldsymbol{x_0}] \cdot [\boldsymbol{x_2} + \boldsymbol{x_0}]/2 \end{bmatrix} = \boldsymbol{A}^{-1}\boldsymbol{B} \tag{2.96}$$

Hence, by using the relationships for the derivative of matrices from Minka [66], we have the following,

$$d(\boldsymbol{AB}) = d\boldsymbol{AB} + \boldsymbol{A}d\boldsymbol{B} \tag{2.97}$$

$$d(\boldsymbol{A}^{-1}) = -\boldsymbol{A}^{-1}(d\boldsymbol{A})\boldsymbol{A}^{-1}, \tag{2.98}$$

where,

$$\begin{aligned} d\boldsymbol{C} &= d(\boldsymbol{A}^{-1}\boldsymbol{B}) = (d\boldsymbol{A}^{-1})\boldsymbol{B} + \boldsymbol{A}^{-1}d\boldsymbol{B} \\ &= -\boldsymbol{A}^{-1}(d\boldsymbol{A})\boldsymbol{A}^{-1}\boldsymbol{B} + \boldsymbol{A}^{-1}d\boldsymbol{B} \\ &= \boldsymbol{A}^{-1}(d\boldsymbol{B} - (d\boldsymbol{A})\boldsymbol{C}), \end{aligned} \tag{2.99}$$

and,

$$\frac{d\boldsymbol{C_o}}{d\boldsymbol{x_0}} = \boldsymbol{A}^{-1} \left(\frac{d\boldsymbol{B}}{dx_0} - \left(\frac{d\boldsymbol{A}}{dx_0} \right) \boldsymbol{C_o} \right), \tag{2.100}$$

where,

$$\frac{d\boldsymbol{A}}{d\boldsymbol{x_0}} = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \quad , \quad \frac{d\boldsymbol{B}}{d\boldsymbol{x_0}} = \frac{1}{2} \begin{bmatrix} -2\boldsymbol{x_0} \\ -2\boldsymbol{x_0}. \end{bmatrix} \tag{2.101}$$

Finally,

$$\frac{d\boldsymbol{C_o}}{d\boldsymbol{x_0}} = \begin{bmatrix} [\boldsymbol{x_1} - \boldsymbol{x_0}]^T \\ [\boldsymbol{x_2} - \boldsymbol{x_0}]^T \end{bmatrix}^{-1} \begin{bmatrix} [\boldsymbol{C_o} - \boldsymbol{x_0}]^T \\ [\boldsymbol{C_o} - \boldsymbol{x_0}]^T \end{bmatrix} \tag{2.102}$$

Now, in the case where the vertex point is clipped at the boundary, the following formula can be used instead which is derived by Baudouin et al. in [67],

$$\frac{d\boldsymbol{C_o}}{d\boldsymbol{x_0}} = \begin{bmatrix} [\boldsymbol{x_1} - \boldsymbol{x_0}]^T \\ [N]^T \end{bmatrix}^{-1} \begin{bmatrix} [\boldsymbol{C_o} - \boldsymbol{x_0}]^T \\ 0 \end{bmatrix}, \tag{2.103}$$

where N is the normal vector at the boundary line segment as shown in f_4 in Figure 2.24. However, this as a consequence is only an approximation of the derivative at the boundary since the Voronoi Cell is clipped (i.e. the Voronoi Cell is not a full cell, but a broken cell that is dictated by the shape of the boundary). Figure 2.26 shows an example of a Voronoi diagram with clipped boundary cells (i.e. red points that are located at a corner on the boundary where the cells are rectangular).

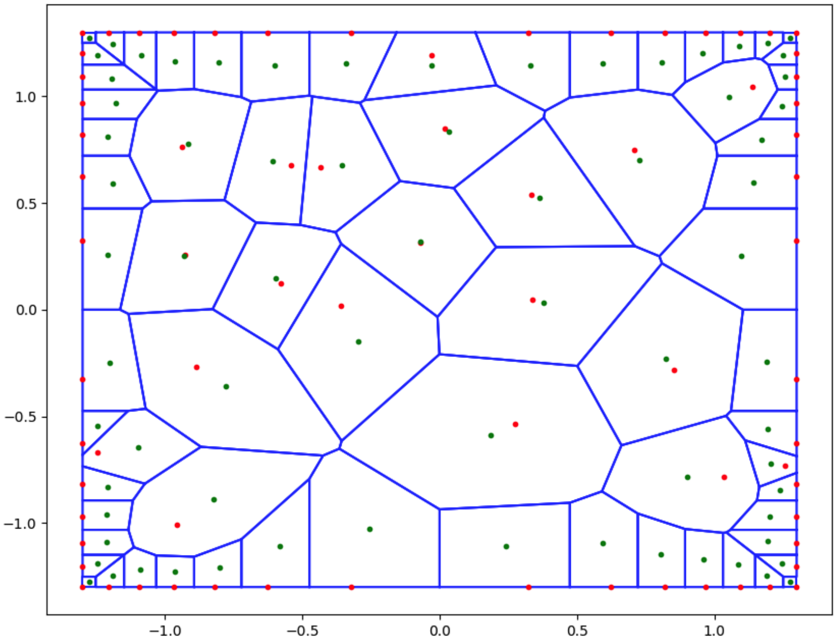


Figure 2.26: Example of Voronoi Diagram with clipped boundary

2.8.3 Quasi-Unit Mesh

Recall that in the general case, the goal of metric-based mesh adaptation is to generate a unit mesh in the metric space. However, this is not guaranteed in practical application as the Riemannian metric field may very often not be compatible with the domain size [18]. Instead, a quasi-unit mesh is targeted for the minimization problem and a relaxed criteria is applied to the required length of the edges on all elements of the mesh.

This is applied through the insertion and removal of Voronoi sites in the domain such that,

$$l_{\mathcal{M}}(\boldsymbol{e_i}) \in [l_{min}, l_{max}] \quad \forall i \in [1,3] \tag{2.104}$$

Where the edge length, $\boldsymbol{e_i}$, is computed using equation 2.68 and i are the indices indicating the edges of a Delaunay triangle.

Figure 2.28 shows an example of edge splitting where the edges that are greater than l_{max} are identified and highlighted in green on the left (i.e. $l_{i,1}, l_{i,2}, l_{i,3}$). Then, a Voronoi site is inserted at the midpoint of it to split the edge by two (i.e. the blue points). Finally, a new Voronoi diagram is generated based on the new set of Voronoi sites shown on the right and three new Voronoi cells are generated (i.e. turquoise, purple, and yellow cells).

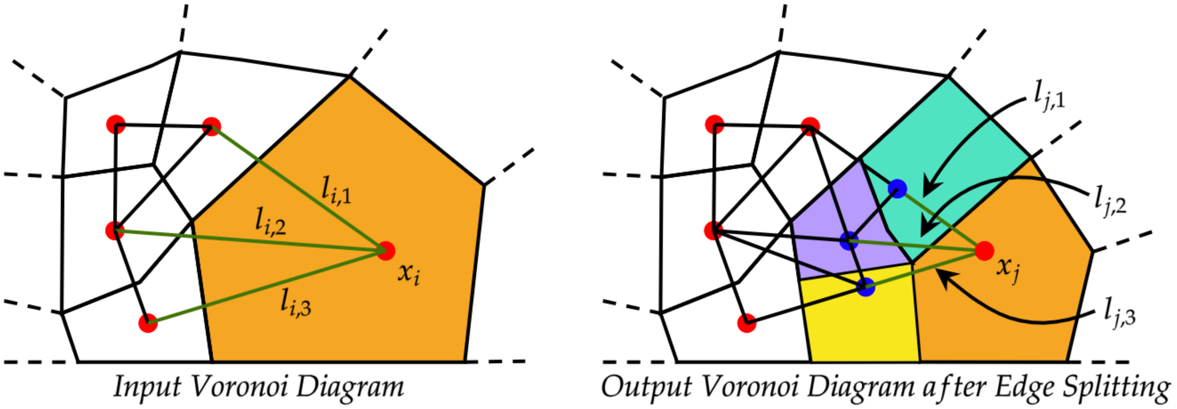


Figure 2.27: Example of Edge Splitting

Similarly, Figure 2.28 shows an example of edge merging or edge removal where the edges that are less than l_{min} are identified and highlighted in orange on the left (i.e. $l_{ij,1}$). Then, this edge is collapsed by merging both x_i and x_j together to form x_k located at the midpoint of $l_{ij,1}$. Finally, a new Voronoi diagram is generated based on the new set of Voronoi sites shown on the right where two Voronoi cells (i.e. yellow cell) are merged together to form a single Voronoi cell with site x_k (i.e. turquoise cell).

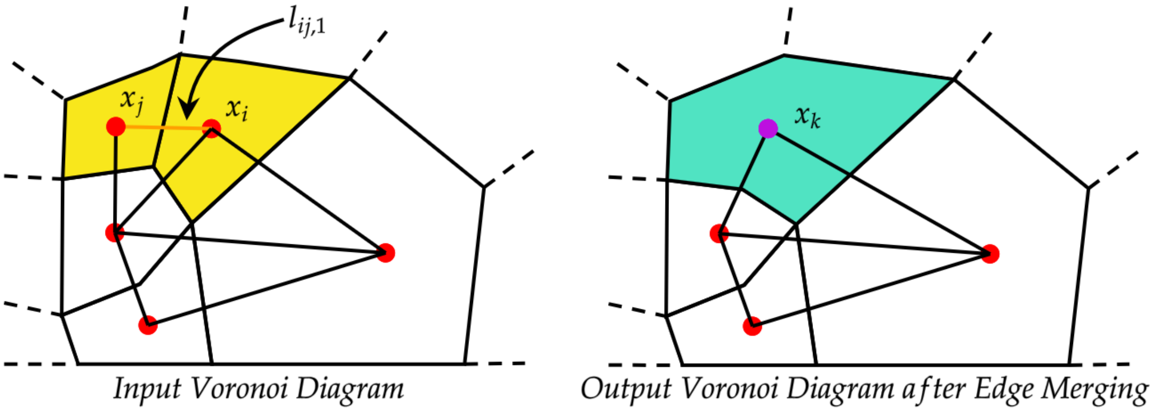


Figure 2.28: Example of Edge Merging

2.9 Simplified Boundary Reconstruction of L_p-CVT

One limitation to the L_p-CVT algorithm is that the generated Voronoi diagram is clipped, which affects the correctness of the gradient computed at the boundary. An improvement to the L_p-CVT method is to consider reconstructing cells at the boundary to yield full Voronoi cells which can help in evaluating the gradient at the boundary as demonstrated by Ekelschot in [68]. This also avoids the need to use equation 2.103 to compute the derivative of the vertex position with respect to the Voronoi sites for a clipped boundary and instead, the regular equation 2.102 can be used.

The approach used in this work is more simpler than what has been presented by Ekelschot. In his work, ghost cells are generated at the boundary to reconstruct the boundary cells (i.e. each boundary cell would have a corresponding ghost cell). The ghost cells are located at a unit metric length (i.e. $l_{\mathcal{M}} = 1$) from the boundary cell and they are assumed to be rectangular because, at a high L_p norm, the Voronoi cells would naturally contain right angles at the minimum state [68]. Instead, our work focuses on generating ghost cells that are equidistant to the initial boundary Γ located at a length of l_r which is the average of the distance $l_{b,i}$ (i.e. edge length in physical space for the i^{th} cell) for all boundary cells located at a unit metric length from the initial boundary, Γ. Then, a ghost boundary Γ' made of ghost cells is generated for reconstructing the boundary cells. Figure 2.29 shows an example of boundary reconstruction. The length l_r is computed as follows,

$$l_r = \frac{\sum_i^{n_b} (l_{b,i})|_{l_{\mathcal{M}}=1}}{n_b}, \tag{2.105}$$

where n_b is the total number of boundary Voronoi cells.

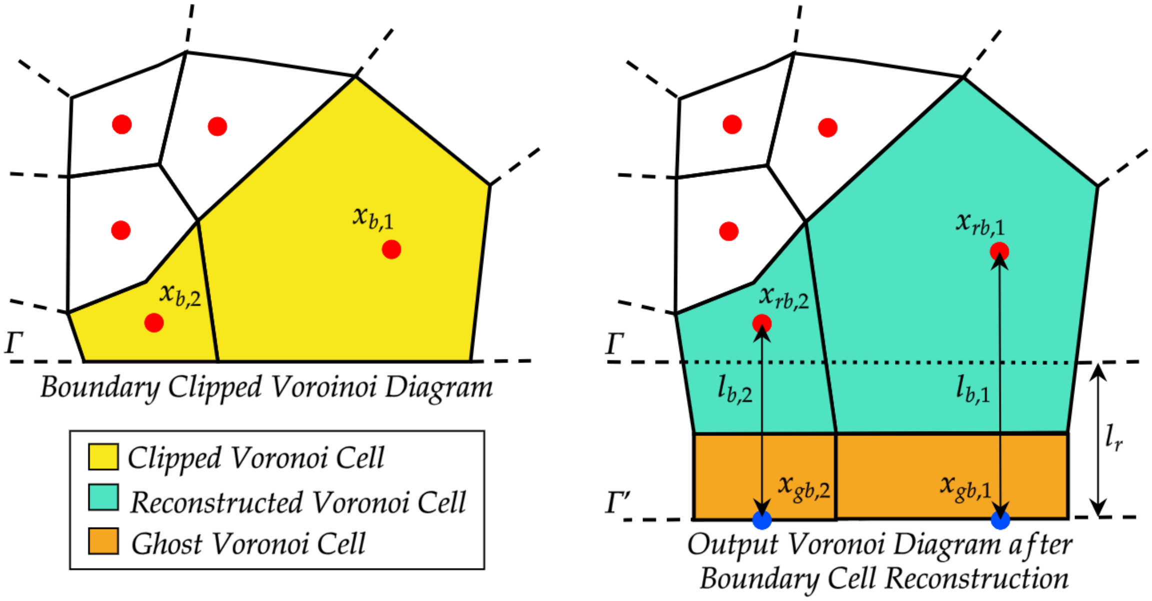


Figure 2.29: Reconstruction of Boundary Voronoi Cell

2.10 Boundary Nodes Distribution

This section discusses the boundary node distribution based on the underlying metric or frame field. For the static boundary case, the boundary nodes must first be placed with spacing between each point that conforms with the underlying metric field for the algorithm to converge. This is because the static boundary case only optimizes the volume points based on the metric field, and therefore, the boundary nodes must be placed close to the optimal point beforehand for the volume points to adapt accordingly and yield an optimal solution. This is not true for the moving boundary case as the boundary points are optimized simultaneously with the volume points which adds robustness to the algorithm. As a result, the output mesh of the moving boundary case is not limited by the initial distribution of the boundary points.

The placement of the boundary nodes according to the underlying metric field is based on the work of Maclean [39]. First, let the boundary curve be parametrized by an equation $\gamma(t)$ for $t \in [0, 1]$ where $\gamma'(t)$ is the tangent vector of $\gamma(t)$. Then, the boundary edge can be defined by an integral of a single vector field based on the frame field in the L_2 norm defined as,

$$L_M(\gamma) = \int_0^1 \|M(\gamma(t))\gamma'(t)\|_2 \, dt = \int_0^1 dL(t) \, dt \tag{2.106}$$

This equation can then be evaluated by the Trapezoidal rule as follows,

$$L_M(\gamma) \approx \sum_{k=0}^{N-1} \frac{1}{2}(dL(t_{k+1}) + dL(t_k)(t_{k+1} - t_k), \tag{2.107}$$

where t_k are the sampled points defined by $t_k = k/N \, \forall k \in [0, N+1]$ where $N+1$ is the total number of points on the boundary including the endpoint. Once the boundary edge length is obtained, points are then added one by one through a partial sum ensuring each added point is of unit length forming the entire boundary. This requires solving an incomplete integral to find the location of each added point by determining t_k as summarized in [39].

2.11 Moving Boundary Formulation for L_p-CVT

A major contribution of this work is to introduce the notion of moving boundary for the L_p-CVT algorithm. The formulation of the moving boundaries has different implications for the code which will be summarized in this section.

2.11.1 Modified Total Energy and Derivative Computation

For the total energy computation, the equation 2.83 can still be used, however, since the boundary cells are now part of the optimization process, they must be added to the equation as well. Previously, with static boundaries, only the volume cells are taken into account for the total energy computation since only the volume Voronoi sites are optimized.

For the total energy derivative computation, a chain rule is applied to equation 2.84 which is similar to what has been shown in [69],

$$\frac{\partial E}{\partial \boldsymbol{x_i}} = \frac{\partial E}{\partial \boldsymbol{x_T}} \frac{\partial \boldsymbol{x_T}}{\partial \boldsymbol{x_i}}, \tag{2.108}$$

where $\frac{\partial E}{\partial \boldsymbol{x_i}}$ includes the boundary Voronoi sites in addition to the volume Voronoi sites.

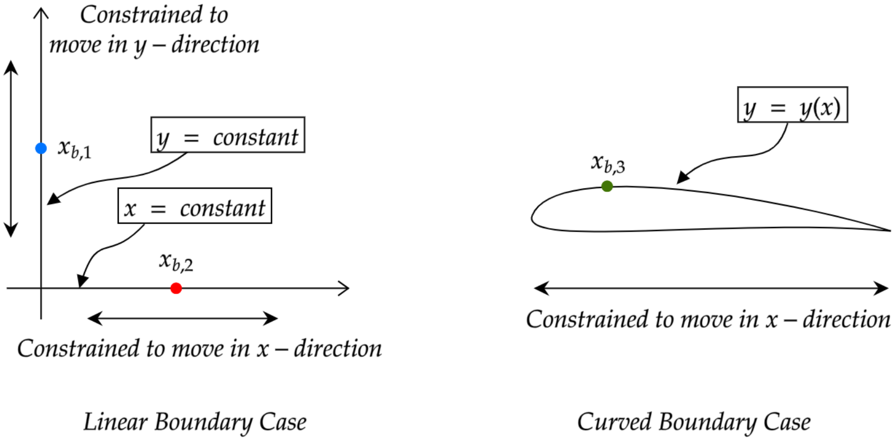


Figure 2.30: Moving boundary formulation for L_p-CVT

One observation to $\frac{\partial E}{\partial \boldsymbol{x_i}}$ is that depending on the boundary definition (i.e. linear or curved boundary), the gradient may be formulated differently. Figure 4.2 shows an example of a linear and curved boundary case.

For the linear boundary case, only the direction of interest will be accounted for in the gradient computation since the boundary Voronoi site may be located on a vertical side or a horizontal side of the boundary (assuming a rectangular domain). Therefore, the design chain-ruled derivative matrix $\frac{\partial \boldsymbol{x_T}}{\partial \boldsymbol{x_i}}$ will contain 0 and 1 depending on the $\frac{\partial E}{\partial \boldsymbol{x_i}}$ that is of interest. An example of gradient computation for both the constrained x-direction and y-direction points in Figure 4.2 is shown as follows,

$$\underbrace{\frac{\partial E}{\partial \boldsymbol{x_i}}}_{1\,\mathrm{x}\,n_D} = \underbrace{\begin{bmatrix} \frac{\partial E}{\partial x_{b,1}} \\ \frac{\partial E}{\partial y_{b,1}} \\ \frac{\partial E}{\partial x_{b,2}} \\ \frac{\partial E}{\partial y_{b,2}} \end{bmatrix}^T}_{1\,\mathrm{x}\,n_T} \underbrace{\begin{bmatrix} \frac{\partial x_{b,1}}{\partial y_{b,1}} & \frac{\partial x_{b,1}}{\partial x_{b,2}} \\ \frac{\partial y_{b,1}}{\partial y_{b,1}} & \frac{\partial y_{b,1}}{\partial x_{b,2}} \\ \frac{\partial x_{b,2}}{\partial y_{b,1}} & \frac{\partial x_{b,2}}{\partial x_{b,2}} \\ \frac{\partial y_{b,1}}{\partial y_{b,1}} & \frac{\partial y_{b,1}}{\partial x_{b,2}} \end{bmatrix}}_{n_T\,\mathrm{x}\,n_D} = \begin{bmatrix} \frac{\partial E}{\partial x_{b,1}} \\ \frac{\partial E}{\partial y_{b,1}} \\ \frac{\partial E}{\partial x_{b,2}} \\ \frac{\partial E}{\partial y_{b,2}} \end{bmatrix}^T \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial y_{b,1}} \\ \frac{\partial E}{\partial x_{b,2}} \end{bmatrix}, \tag{2.109}$$

where n_T is the total number of variables and n_D is the total number of design variables. For the curved boundary case, the exact equation of the boundary can be utilized for the gradient computation while assuming that the x-direction is of interest. This is because once the new update on the x position is received, the y value can be simply updated through the boundary equation, $y = y(x)$. An example of gradient computation for the constrained x-direction points on the curved boundary in Figure 4.2 is shown as follows,

$$\underbrace{\frac{\partial E}{\partial \boldsymbol{x_i}}}_{1\,\mathrm{x}\,n_D} = \underbrace{\begin{bmatrix} \frac{\partial E}{\partial x_{b,3}} \\ \frac{\partial E}{\partial y_{b,3}} \end{bmatrix}^T}_{1\,\mathrm{x}\,n_T} \underbrace{\begin{bmatrix} \frac{\partial x_{b,3}}{\partial x_{b,3}} \\ \frac{\partial y_{b,3}}{\partial x_{b,3}} \end{bmatrix}}_{n_T\,\mathrm{x}\,n_D} = \begin{bmatrix} \frac{\partial E}{\partial x_{b,3}} \\ \frac{\partial E}{\partial y_{b,3}} \end{bmatrix}^T \begin{bmatrix} 1 \\ \frac{\partial y_{b,3}}{\partial x_{b,3}} \end{bmatrix} = \frac{\partial E}{\partial x_{b,3}} + \frac{\partial y_{b,3}}{\partial x_{b,3}}, \tag{2.110}$$

where $y_{b,3}$ is equal to the boundary equation, $y = y(x)$ and $\frac{\partial y_{b,3}}{\partial x_{b,3}}$ is equal to the derivative of $y(x)$.

2.12 All-Quad Mesh Generator

Once the L_p-CVT algorithm has converged, the geometric duality of the Voronoi diagram is generated which results in a Delaunay Triangulation, or a mesh full of triangles. This triangular mesh is generated through the CGAL library [70]. Since the L_p-CVT algorithm is solved using a high p norm of six as mentioned in section 2.6, the output mesh is expected, but not guaranteed, to contain right-angled triangles which facilitate the process of recombination of triangles to form quadrilaterals. Therefore, a refinement procedure is performed by adding additional points at the midpoint of each edge to increase the likelihood of a quad-dominant mesh after the recombination step. The recombination process is done through the Blossom-Quad algorithm from the GMSH library [42, 71]. Once the quad-mesh is generated, the mesh is coded into a .MSH file (which is a GMSH format) for compatibility with our flow solver PHILIP [72].

2.13 Overview of L_p-CVT algorithm

Solving the L_p-CVT problem requires a minimization of the functional, equation 2.83. The goal is to reach a minimum state where the Voronoi sites x_i are aligned with the centroid of the Voronoi cell. In other words, the red points which are the x_i must align on top of the green points which are the centroid of the Voronoi cell in Figure 2.26. An example of this has also been shown previously in Figure 2.8. With the formulation of the gradient from equation 2.84, the functional can be solved using a quasi-Newton method. For this work, the L-BFGS algorithm is employed from the ROL library of Trilinos [73] to solve the minimization problem. Note that convergence in the algorithm is achieved once the norm of the step size or the gradient as shown in equation 2.84 has reached below a certain tolerance. Additionally, edge splitting and merging are performed as an outer loop to the L_p-CVT algorithm until approximately 2% of the total edges are flagged for splitting and merging.

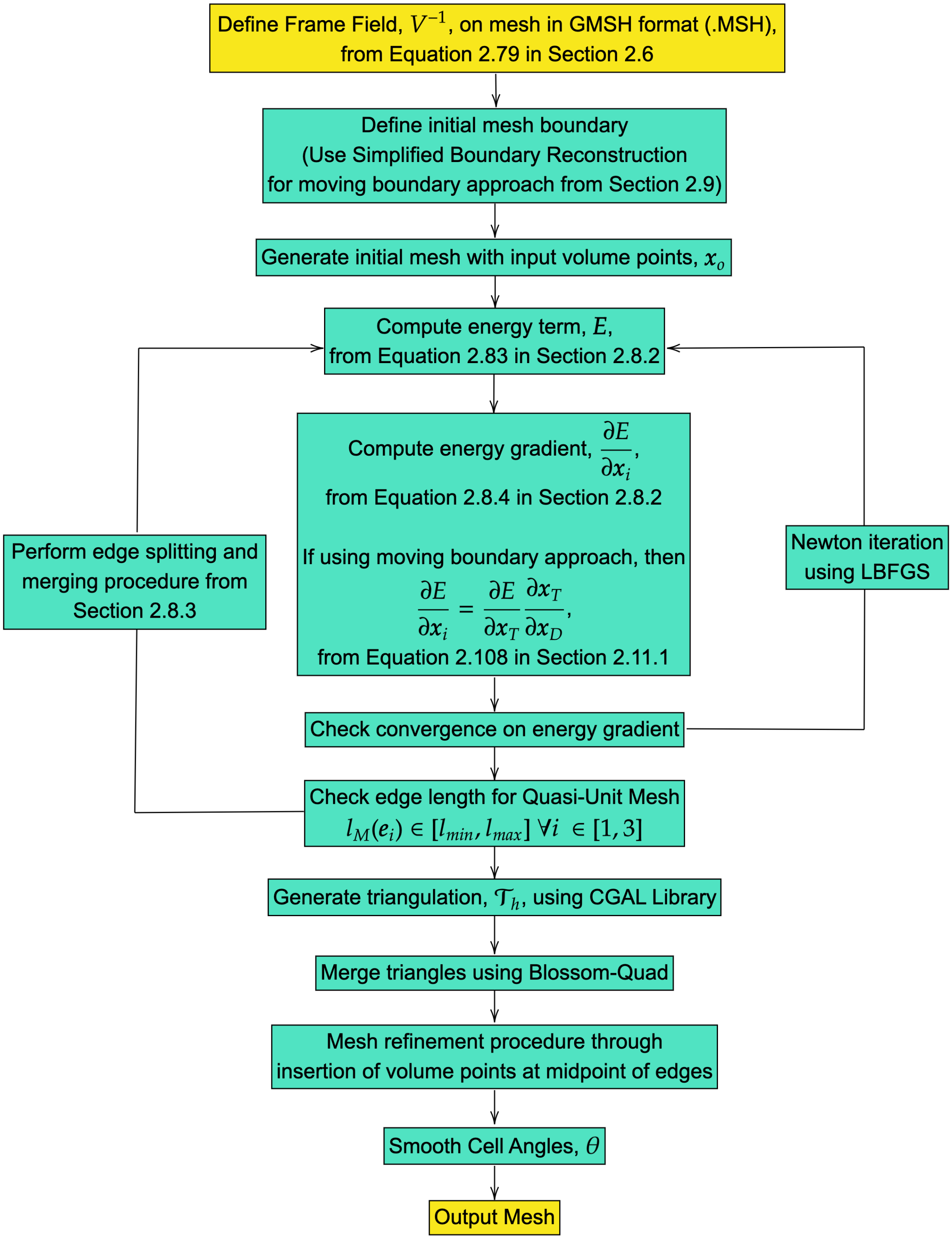

Define Frame Field, V^{-1}, on mesh in GMSH format (.MSH), from Equation 2.79 in Section 2.6
Define initial mesh boundary (Use Simplified Boundary Reconstruction for moving boundary approach from Section 2.9)
Generate initial mesh with input volume points, x_o
Compute energy term, E, from Equation 2.83 in Section 2.8.2
Compute energy gradient, $\frac{\partial E}{\partial x_i}$, from Equation 2.8.4 in Section 2.8.2
If using moving boundary approach, then $\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial x_T}\frac{\partial x_T}{\partial x_D}$, from Equation 2.108 in Section 2.11.1
Perform edge splitting and merging procedure from Section 2.8.3
Newton iteration using LBFGS
Check convergence on energy gradient
Check edge length for Quasi-Unit Mesh $l_M(e_i) \in [l_{min}, l_{max}]\ \forall i \in [1,3]$
Generate triangulation, $\mathcal{T}_h$, using CGAL Library
Merge triangles using Blossom-Quad
Mesh refinement procedure through insertion of volume points at midpoint of edges
Smooth Cell Angles, θ
Output Mesh

Figure 2.31: Flowchart showing the L_p-CVT algorithm

Chapter 3

Numerical Methods

A brief introduction to the discontinuous Galerkin method will be presented in this section which is the numerical scheme employed in our in-house flow solver PHiLiP [72]. Then, a brief introduction to mesh adaptation will serve as the framework for the analytical as well as discrete metric test case.

High-order numerical schemes are of interest as they present a significantly better cost of computation without sacrificing accuracy [74]. In high-order methods, the solution can be approximated using basis functions with polynomials of degree "p" on a mesh with a spatial discretization of size "h". Hence, high-order schemes possess the flexibility of allowing the solution to be represented on various configurations of "hp" mesh thereby allowing the possibility of adaptation techniques to be developed on this basis. For this work, the discontinuous Galerkin method is used for solving Euler's equation in two-dimensional space. This method was originally introduced by Reed and Hill and can be found here [75]. In regards to adaptation, the solution can be adapted either by changing the order of the element to increase the accuracy of the solution by using higher degree polynomials to represent the solution or by changing the solution node placement on the mesh at the location of interest. In this work, the "p" order of the solution is assumed to be uniform throughout the domain.

3.1 Discontinuous Galerkin Method

In CFD, the governing equation of interest for modeling fluid motions can be written in the form of a general conservation law known as,

$$\boldsymbol{R}(\boldsymbol{u}) = \frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F}(\boldsymbol{u}) - \boldsymbol{S}(\boldsymbol{u}) = 0, \tag{3.1}$$

where $\boldsymbol{u}$ is the flow solution, $\boldsymbol{R}(\boldsymbol{u})$ is the residual, $\boldsymbol{F}(\boldsymbol{u})$ the flux vector and $\boldsymbol{S}(\boldsymbol{u})$ is the source term. In particular, Euler's equation and the Navier-Stokes equation can be expressed in the form of Equation 3.1.

To solve equation 3.1, consider a computational domain, Ω_h, with a boundary Γ which approximates the actual domain, Ω, with techniques shown in Chapter 2. Then, the discrete solution can be defined as u which is an approximation to the exact solution defined as u in Ω. The principle of the DG method is to represent the discrete solution u_h as a sum of the local discrete solution u_h^k represented on a local domain or cell known as Ω_k where $\Omega_k \in \Omega_h$. Each local domain has a boundary Γ_k and the local discrete solution is represented by,

$$u_h^k(x) = \sum_{i=1}^{N_k(p_k)} u_i^k \phi_{h,i}(x) \quad \forall x \in \Omega_h, \tag{3.2}$$

where p_k is the order of the element, $\phi_{h,i}$ is the i^{th} shape function defined on element k that is defined by a set of basis function $\mathcal{V}_{hp}$ where $\phi_{h,i} \in \mathcal{V}_{hp}$, and $N_k(p_k)$ is the total number of solution points within the element k. Note that $N_k(p_k)$ depends on the type of cell chosen (i.e. triangle or quadrilateral) as well as the order of the element p_k and is defined as,

$$N_k(p_k) = \frac{1}{2}(p_k+1)(p_k+2) \quad \text{(Triangle)} \tag{3.3}$$

$$N_k(p_k) = (p_k+1)^2 \quad \text{(Quadrilateral)} \tag{3.4}$$

More information about high-order elements can be found in Chapter 5 of *The Finite Element Method in Charged Particle Optics* by Khursheed [76]. An example of both the triangle as well as quadrilateral for p_k equals to 2 (i.e. second-order element) is shown in Figure 3.1,

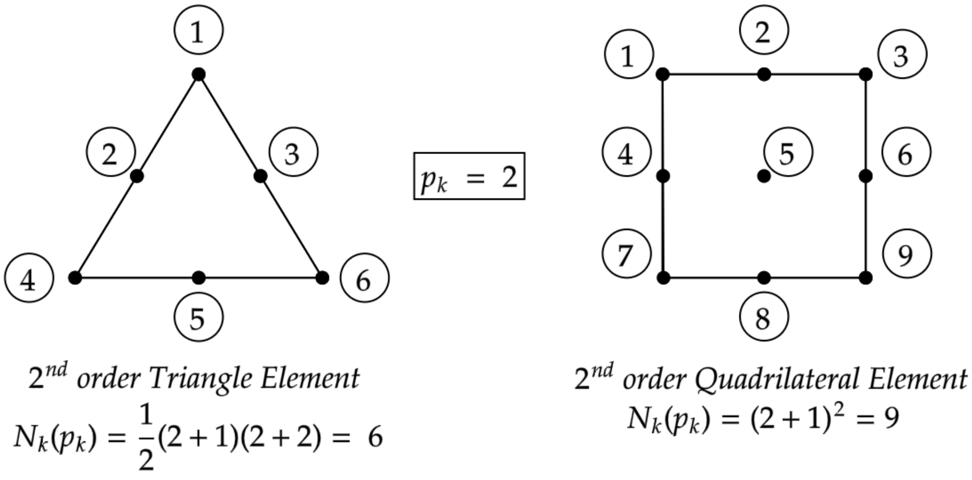

1
2
3
4
5
6
$p_k = 2$
1
2
3
4
5
6
7
8
9
2^{nd} order Triangle Element
$N_k(p_k) = \frac{1}{2}(2+1)(2+2) = 6$
2^{nd} order Quadrilateral Element
$N_k(p_k) = (2+1)^2 = 9$

Figure 3.1: 2^{nd} order triangle and quadrilateral elements

Now, to represent the solution, a sum of the local solution can be expressed as follows,

$$u_h(x) = \bigoplus_{\Omega_k \in \Omega_h} u_i^k(x). \tag{3.5}$$

From equation 3.1, it is possible to solve for a solution by multiplying the equation with arbitrary test functions $\psi_h \in \mathcal{V}_{hp}$ and requiring that the residual $\boldsymbol{R}(\boldsymbol{u})$ to be orthogonal to the test functions. After integrating by parts, the following is obtained,

$$\int_{\Omega_k} \psi_h \frac{\partial u_h}{\partial t} d\Omega - \int_{\Omega_k} \nabla \psi_h \cdot F(u_h) d\Omega - \int_{\Omega_k} \psi_h S(u_h) d\Omega + \int_{\partial \Omega_k} \psi_h \hat{F}(u_h^+, u_h^-, n) d\Gamma = 0, \tag{3.6}$$

, where n is the normal vector at an element boundary, $\hat{F}$ is the numerical flux that is conserved across the elements, where u_h^+, and u_h^-, are the discontinuous interior and exterior solutions.

3.2 Discrete Error Minimization

The premise of metric-based mesh adaptation is to adapt the mesh in accordance with a metric tensor that encodes the targeted mesh to increase the accuracy of the solution. As such, the error in the solution is known as the discrete error which is defined as follows,

$$E_h(\boldsymbol{u}_h) = \|\boldsymbol{u} - \boldsymbol{u}_h\|^q_{L^q(\Omega_h)} = \int_{\Omega_h} (\boldsymbol{u} - \boldsymbol{u}_h)^q dx, \tag{3.7}$$

where the discrete error is evaluated with respect to an L_q norm, $\boldsymbol{u}$ is the exact solution and $\boldsymbol{u}_h$ is the discrete solution.

The goal is then to optimize the mesh while attempting to reduce the discrete error presented by equation 3.7. At the same time, it is also desired to generate an adapted mesh with the optimal number of degrees of freedom (DOFs) to not sacrifice accuracy for higher computational cost. Therefore, the global discrete error minimization problem is formulated as follows from [16].

Given the existence of the exact solution $u \in C^\infty(\Omega)$ and a maximum number of degrees of freedom C, or target complexity, solve,

$$\min_{Q_{hp}} E_h(\Pi_{hp}\boldsymbol{u}) = \|\boldsymbol{u} - \Pi_{hp}\boldsymbol{u}\|^q_{L^q(\Omega_h)} \tag{3.8}$$

$$\text{s.t.} \quad N_{hp}(Q_{hp}) \leq C,$$

where $N_{hp}(Q_{hp})$ is the total number of DOFs and Π_{hp} is the optimal projection operator for the solution space such that

$$\Pi_{hp}\boldsymbol{u} = \operatorname*{argmin}_{\boldsymbol{u_h} \in \mathcal{V}_{hp}} E_h(\boldsymbol{u_h}) \tag{3.9}$$

where $\mathcal{V}_{hp}$ is the discontinuous hp solution space defined by the set of basis functions on each element.

Solving the problem of equation 3.8 is a difficult task in the discrete setting (i.e. the discrete mesh) as there are considerations of both continuous (i.e. shape of the elements which can vary smoothly) and discrete elements (i.e. total number of elements, element connectivity, etc.) that need to be optimized and taken into account [17, 39, 77]. Instead, a fully continuous framework is of interest which will be discussed in Chapter 4.

3.3 Adjoint Method

In CFD, it is often of interest to solve the general conservation law and converge on a particular parameter with minimal error. For instance, we might be interested in the lift or drag value which can computed as a functional of interest. Then, the solution can aim to minimize the error on this functional by adapting the mesh and to do this, the adjoint method is often used.

Consider a functional of the flow solution $\mathcal{J}(u, x)$ where $(u, x) \in \mathbb{N}^n$, and also a constraint where $\mathcal{R}(u, x) = 0$ denotes the solution residual as stated in equation 3.1 and its partial derivative $\mathcal{R}_x$ is nonsingular everywhere in the domain Ω. Then, we are interested in minimizing $\mathcal{J}$ by identifying the critical points through setting $\mathrm{d}_x\mathcal{J}$ to zero where $\mathrm{d}_x\mathcal{J}$ defines the change in the functional with respect to the design parameter x. The total derivative $\mathrm{d}_x\mathcal{J}$ can be formulated as follows,

$$\mathrm{d}_x\mathcal{J} = \mathrm{d}_x\mathcal{J}(u, x) = \partial_u\mathcal{J}\mathrm{d}_x u + \partial_x\mathcal{J} = \frac{\partial\mathcal{J}}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial\mathcal{J}}{\partial x} \tag{3.10}$$

And likewise, the constraint can be defined as,

$$\mathrm{d}_x\mathcal{R}(u, x) = 0 \quad \text{since, } \mathcal{R}(u, x) = 0 \text{ everywhere in } \Omega \tag{3.11}$$

Therefore,

$$\mathrm{d}_x\mathcal{R} = \mathrm{d}_x\mathcal{R}(u, x) = \partial_u\mathcal{R}\mathrm{d}_x u + \partial_x\mathcal{R} = \frac{\partial\mathcal{R}}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial\mathcal{R}}{\partial x} = 0 \tag{3.12}$$

And,

$$\frac{\partial u}{\partial x} = -\frac{\partial\mathcal{R}}{\partial x}\left(\frac{\partial\mathcal{R}}{\partial u}\right)^{-1} \tag{3.13}$$

Now, substituting equation 3.13 into 3.10, we obtain the following where we can also introduce the parameter ψ^T which is known as the adjoint variable,

$$\begin{aligned} \mathrm{d}_x\mathcal{J} &= \underbrace{-\frac{\partial \mathcal{J}}{\partial u}\left(\frac{\partial \mathcal{R}}{\partial u}\right)^{-1}}_{\psi^T}\frac{\partial \mathcal{R}}{\partial x} + \frac{\partial \mathcal{J}}{\partial x} \\ &= \frac{\partial \mathcal{J}}{\partial x} - \psi^T\frac{\partial \mathcal{R}}{\partial x} \end{aligned} \tag{3.14}$$

Another derivation of the same result can be done through the Lagrangian as follows where ψ are the Lagrange multipliers or the adjoint variable,

$$\mathcal{L}(u,x,\psi) = \mathcal{J}(u,x) - \psi^T\cancelto{0}{\mathcal{R}(u,x)}\text{since, } \mathcal{R}(u,x) = 0 \text{ everywhere in } \Omega \tag{3.15}$$

Hence, we may compute the total derivative of $\mathrm{d}_x\mathcal{J}(u,x)$ as $\mathrm{d}_x\mathcal{L}(u,x,\psi)$ instead due to the relationship in equation 3.15 and also, note that $\mathcal{R}(u,x) = 0$ everywhere in the domain Ω, then,

$$\begin{aligned} \mathrm{d}_x\mathcal{L}(u,x,\psi) &= \mathrm{d}_x\mathcal{J}(u,x) - \mathrm{d}_x(\psi^T\mathcal{R}(u,x)) \\ &= \mathrm{d}_x\mathcal{J}(u,x) + \mathrm{d}_x\psi^T\cancelto{0}{\mathcal{R}(u,x)} + \psi^T\mathrm{d}_x\mathcal{R}(u,x) \\ &= \frac{\partial \mathcal{J}}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial \mathcal{J}}{\partial x} - \psi^T\left(\frac{\partial \mathcal{R}}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial \mathcal{R}}{\partial x}\right) \\ &= \frac{\partial u}{\partial x}\underbrace{\left(\frac{\partial \mathcal{J}}{\partial u} - \psi^T\frac{\partial \mathcal{R}}{\partial u}\right)}_{\text{adjoint equation}} + \left(\frac{\partial \mathcal{J}}{\partial x} - \psi^T\frac{\partial \mathcal{R}}{\partial x}\right) \end{aligned} \tag{3.16}$$

In both equations 3.14 and 3.16, we can recover the total derivative $\mathrm{d}_x\mathcal{J} = \frac{\partial \mathcal{J}}{\partial x} - \psi^T\frac{\partial \mathcal{R}}{\partial x} \approx 0$ through solving the adjoint equation,

$$\frac{\partial \mathcal{J}}{\partial u} - \psi^T\frac{\partial \mathcal{R}}{\partial u} = 0 \tag{3.17}$$

The benefit of this formulation is evident in equation 3.14 where the multiplication $\frac{\partial \mathcal{R}}{\partial x}\left(\frac{\partial \mathcal{R}}{\partial u}\right)^{-1}$ is required N times for N design parameters which is computationally expensive. However, through solving the adjoint equation and finding the adjoint variable ψ^T, this process can be performed one time through solving a single linear system of equation (equation 3.26) for N design parameters. Lastly, the functional of the flow solution can be expressed in an integral form such as,

$$\mathcal{J}(u) = \int_{\Omega} g_{\Omega}(u) d\Omega + \int_{\Gamma} g_{\Gamma}(u) d\Gamma, \tag{3.18}$$

where equation 3.18 is the general integral form that allows for the computation of parameters such as lift and drag value.

3.3.1 Dual-Weight Residual

To apply the concept of the adjoint method in mesh adaptation, we can adopt the dual-weight residual (DWR) method which was first attempted by Venditti and Darmofal [36, 37, 38]. This approach is also known as goal-oriented mesh adaptation which utilizes the concept of fine and coarse grids.

First, a projection operator must be introduced to traverse from the coarse grid to the fine grid and vice versa which is defined as follows,

$$u_h^H = I_h^H u_H, \tag{3.19}$$

where h represents the fine grid, and H represents the coarse grid. Therefore, u_h^H is the fine grid solution resulting from a projection of the coarse grid solution u_H. The idea of the DWR method is to utilize the fine grid solution for accurately estimating the solution represented on the coarse grid while not ever requiring the solution to converge on the fine grid. Hence, we are interested in $\mathcal{J}_h(u_h)$ which can be, first, expanded through a Taylor series expansion with the higher order terms neglected as,

$$\mathcal{J}_h(u_h) \approx \mathcal{J}_h(u_h^H) + \left.\frac{\partial \mathcal{J}_h}{\partial u_h}\right|_{u_h^H} (u_h - u_h^H). \tag{3.20}$$

Similarly, the residual can also be expressed as,

$$R_h(u_h) = 0 \approx R_h(u_h^H) + \left.\frac{\partial R_h}{\partial u_h}\right|_{u_h^H} (u_h - u_h^H). \tag{3.21}$$

Assuming that the fine grid solution converges and the residual is equal to zero, the discrete error between the actual fine grid solution and its projected solution from the coarse grid solution can be approximated as follows,

$$(u_h - u_h^H) \approx \left(\left.\frac{\partial R_h}{\partial u_h}\right|_{u_h^H} \right)^{-1} R_h(u_h^H). \tag{3.22}$$

Now, 3.22 can be substituted into equation 3.20 to yield the following,

$$\mathcal{J}_h(u_h) \approx \mathcal{J}_h(u_h^H) - \underbrace{\left.\frac{\partial \mathcal{J}_h}{\partial u_h}\right|_{u_h^H} \left(\left.\frac{\partial R_h}{\partial u_h}\right|_{u_h^H} \right)^{-1}}_{\left(\psi_h|_{u_h^H}\right)^T} R_h(u_h^H), \tag{3.23}$$

where $\psi_h|_{u_h^H}^T$ satisfies the fine grid discrete adjoint equation,

$$\left.\frac{\partial \mathcal{J}_h}{\partial u_h}\right|_{u_h^H} - \left(\psi_h|_{u_h^H}\right)^T \left.\frac{\partial R_h}{\partial u_h}\right|_{u_h^H} = 0. \tag{3.24}$$

And also, the fine grid discrete adjoint solution is evaluated from the coarse grid through a projection operator similar to equation 3.25,

$$\psi_h^H = I_h^H \psi_H, \tag{3.25}$$

where ψ_H^T satisfies the coarse grid discrete adjoint equation,

$$\frac{\partial \mathcal{J}_H}{\partial u_H} - \psi_H^T \frac{\partial R_H}{\partial u_H} = 0. \tag{3.26}$$

Finally, the discrete error between the actual fine grid functional value and its projected value from the coarse grid solution can be approximated from equation 3.27 as follows,

$$\mathcal{J}_h(u_h) - \mathcal{J}_h(u_h^H) \approx - \left(\psi_h|_{u_h^H} \right)^T R_h(u_h^H). \tag{3.27}$$

In this work, this discrete error term will be approximated through the sum of cell-wise DWR values, namely,

$$\left| \mathcal{J}_h(u_h) - \mathcal{J}_h(u_h^H) \right| \leq \sum_k \left| \left(\psi_h|_{u_h^H} \right)^T R_h(u_h^H) \right|, \tag{3.28}$$

where the equation 3.28 acts as a bound on the functional error and reducing the cell-wise DWR value locally will globally improve the error bound (through the sum).

Chapter 4

Continuous Mesh Model and Error Estimates

In section 3.2, the global discrete minimization problem is presented, however, it is not easily solvable through a discrete framework. Therefore, a continuous mesh model will be presented in this section which stems from the original work of Loseille and Alauzet [18, 19].

4.1 Continuous Mesh Model

The main idea behind the continuous mesh model is the duality between the discrete mesh that can be generated through techniques presented in Chapter 2 and the Riemannian metric space as discussed in Chapter 3. Most notably, a continuous framework is defined as a collection of continuous elements which are known as metric tensors $\mathcal{M}$ which are defined in the domain Ω such that $M = \mathcal{M}(x)\,\forall x \in \Omega$ where $\mathcal{M}(x)$ is known as a Riemannian metric space [18]. Through the introduction of the continuous mesh model, it is possible to introduce a continuous error model where the optimal mesh can be solved through the use of variational calculus. For the scope of this work, frame fields will be considered as the underlying continuous mesh model.

4.1.1 Continuous Element Definition of Area and Density

From section 2.6, the notion of the orientation (i.e. $R(\theta)$), size (h), and anisotropy (ρ), has been established for frame fields. Using these parameters, the area as well as the density at any point, x, in the domain, Ω, can be computed as follows,

$$A(\boldsymbol{x}) = 4 \cdot h(\boldsymbol{x})^2 = 4 \cdot \left(\sqrt{h_1(\boldsymbol{x}) \cdot h_2(\boldsymbol{x})}\right)^2 = 4 \cdot \det\bigl(V(\boldsymbol{x})\bigr)^{-1} \quad \text{(Area)} \tag{4.1}$$

$$d(\boldsymbol{x}) = \frac{1}{A(\boldsymbol{x})} = \frac{1}{4 \cdot \det\bigl(V(\boldsymbol{x})\bigr)} \quad \text{(Density)} \tag{4.2}$$

Equation 4.1 defines the area of a parallelogram which matches the definition of a frame field where h_1 and h_2 represent half of the height and base of the parallelogram.

4.1.2 Continuous Mesh Characterization

From [17], the continuous mesh model can be characterized as follows,

- Orientation is equivalent to the rotation matrix $R(\theta)$ where $\theta \in [0, \pi)$
- Stretching is equivalent to the anisotropy $\rho(h_1, h_2) \in (0, 1]$
- Size is equivalent to the density $d(x) > 0$

These three parameters form a tuplet which characterizes the continuous element. In addition, we can characterize the continuous mesh by approximating the total number of cells, N, as well as the total number of degrees of freedom (also known as continuous target complexity), $\mathcal{N}$. To achieve this, we can simply integrate the density over the domain and multiply it by the polynomial degree of each element to yield respectively,

$$N(V) = \int_\Omega d(\boldsymbol{x}) d\boldsymbol{x} = \int_\Omega \frac{1}{4 \cdot \det\bigl(V(\boldsymbol{x})\bigr)} d\boldsymbol{x} \quad \text{(Total number of cells)}, \tag{4.3}$$

$$\mathcal{N}(V, \mathcal{P}) = \int_\Omega d(\boldsymbol{x})(\mathcal{P}(\boldsymbol{x}) + 1)^2 d\boldsymbol{x} \quad \text{(Total number of degrees of freedom)}, \tag{4.4}$$

where $\mathcal{P}(\boldsymbol{x})$ is equal to the distribution of polynomial degree across the domain.

4.2 Continuous Error Model

This section will discuss how to encode the discrete error from the solution onto the frame field (i.e. continuous element) and solve the global discretization problem as shown in equation 3.8 in a continuous framework. In particular, the goal is to first, optimize the rotation matrix as well as the anisotropy locally on each element, and solve a global problem to find the optimal density in the continuous mesh.

4.2.1 Local Interpolation Error

First, let's derive the local continuous interpolation error based on the continuous mesh model [16]. Consider a smooth function u, a point $\tilde{\boldsymbol{x}} = (\tilde{x_1}, \tilde{x_2})^T \in \Omega$ and a polynominal degree $p \in \mathbb{N}$. Using Taylor series expansion, the function $u(\boldsymbol{x})$ can be expressed around $\tilde{\boldsymbol{x}}$ as follows,

$$u(\boldsymbol{x}) = \sum_{k=0}^{p+1} \frac{1}{k!} \left(\sum_{l=0}^{k} \frac{k!}{l!(k-l)!} \frac{\partial^k u(\tilde{\boldsymbol{x}})}{\partial x_1^l \partial x_2^{k-l}} (x_1 - \tilde{x_1})^l (x_2 - \tilde{x_2})^{k-l} \right) + O(|\boldsymbol{x} - \tilde{\boldsymbol{x}}|^{p+2}) \qquad (4.5)$$

Then, we can also introduce a local projection operator $\Pi_{\tilde{\boldsymbol{x}},p}$ where the partial derivatives for the projected function $\Pi_{\tilde{\boldsymbol{x}},p}u(\tilde{\boldsymbol{x}})$ has the same value as the partial derivatives of u at the point $\tilde{\boldsymbol{x}}$ up to an order of p,

$$\frac{\partial^k}{\partial x_1^2 \partial x_2^{k-l}} \Pi_{\tilde{\boldsymbol{x}},p} u(\tilde{\boldsymbol{x}}) = \frac{\partial^k}{\partial x_1^2 \partial x_2^{k-l}} u(\tilde{\boldsymbol{x}}), \qquad (4.6)$$

where l is equal to any number from 0 to k, and k is equal to any number from 0 up to the polynomial order p.

Hence, it is evident that substituting the solution $u(\tilde{\boldsymbol{x}})$ defined by equation 4.5 into equation 4.6 and taking the difference on both sides yields the following,

$$u(\boldsymbol{x}) - \Pi_{\tilde{\boldsymbol{x}},p} u(\boldsymbol{x}) = C \left(\cancelto{0}{\frac{\partial^k}{\partial x_1^2 \partial x_2^{k-l}} \Pi_{\tilde{\boldsymbol{x}},p} u(\tilde{\boldsymbol{x}}) - \frac{\partial^k}{\partial x_1^2 \partial x_2^{k-l}} u(\tilde{\boldsymbol{x}})} \right) + e^{int}_{\tilde{\boldsymbol{x}},p}(\tilde{\boldsymbol{x}}) + O(|\boldsymbol{x} - \tilde{\boldsymbol{x}}|^{p+2}), \qquad (4.7)$$

where the function $e^{int}_{\tilde{\boldsymbol{x}},p}$ is the local interpolation error of degree p around $\tilde{\boldsymbol{x}}$,

$$e^{int}_{\tilde{\boldsymbol{x}},p}(\boldsymbol{x}) = \frac{1}{(p+1)!}\left(\sum_{l=0}^{k}\binom{p+1}{l}\frac{\partial^{p+1}u(\tilde{\boldsymbol{x}})}{\partial x_1^l \partial x_2^{p+1-l}}(x_1-\tilde{x_1})^l(x_2-\tilde{x_2})^{p+1-l}\right). \tag{4.8}$$

Equation 4.8 is equal to zero if evaluated at the point $\tilde{\boldsymbol{x}}$. Now, the objective is to find a bound to this local interpolation error through the use of the frame field (i.e. continuous element), and this bound can be defined through the directional derivative of $u(\boldsymbol{x})$ where, as demonstrated in [16, 39], the bound is given by,

$$\left|e^{int}_{\tilde{\boldsymbol{x}},p}(\boldsymbol{x})\right| \leq \left((\boldsymbol{x}-\tilde{\boldsymbol{x}})^T R(\psi)\text{diag}(A_1, A_2)^{\frac{2}{p+1}} R(-\psi)(\boldsymbol{x}-\tilde{\boldsymbol{x}})\right) \tag{4.9}$$

Note that $R(\psi)$ is the rotation matrix as defined in 2.77 and A_1 as well as A_2 are the maximum value of the $p+1$ directional derivative in directions $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$ respectively as defined below,

$$A_1(\tilde{\boldsymbol{x}}, p) = \max_{\|\boldsymbol{\xi}\|_2=1} |D^{p+1}_{\boldsymbol{\xi}} u(\tilde{\boldsymbol{x}})| \tag{4.10}$$

$$\boldsymbol{\xi}_1(\tilde{\boldsymbol{x}}, p) = \underset{\|\boldsymbol{\xi}\|_2=1}{\text{argmax}} |D^{p+1}_{\boldsymbol{\xi}} u(\tilde{\boldsymbol{x}})| \tag{4.11}$$

$$\psi(\tilde{\boldsymbol{x}}, p) \in [0, 2\pi) \text{ such that } \boldsymbol{\xi}_1(\tilde{\boldsymbol{x}}, p) = (cos(\psi), sin(\psi)) \tag{4.12}$$

$$A_2(\tilde{\boldsymbol{x}}, p) = |D^{p+1}_{\boldsymbol{\xi_2}} u(\tilde{\boldsymbol{x}})|, \text{ where} \boldsymbol{\xi_1} \cdot \boldsymbol{\xi_2} = 0, \tag{4.13}$$

where the directional derivatives are found from the enriched solution u_h^+ that is $p+1$ times differentiable and recovered through an H^1 patchwise reconstruction of the neighbor solution cells. A similar recovery method can be found in more detail in these two papers by Zienkiewics and Zhu [13, 14]. Note that the patchwise reconstruction is performed using four neighbor solution points shown as,

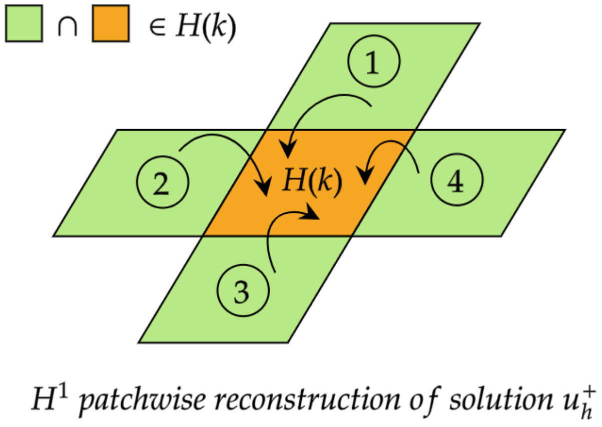


Figure 4.1: H^1 patchwise reconstruction

The enriched solution is therefore recovered by multiplying both the enriched solution and the discrete solution by test functions ϕ^+ which are $p+1$ times differentiable,

$$\langle u_h^+ - u_h, \phi^+ \rangle_{H^1} = 0, \tag{4.14}$$

$$\langle u_h^+, \phi^+ \rangle_{H^1} = \langle u_h, \phi^+ \rangle_{H^1} \quad \forall \phi^+ \in \mathbb{P}^{p+1}, \tag{4.15}$$

where,

$$u_h^+ = \mathbf{\Phi}^+ \boldsymbol{a}, \tag{4.16}$$

$$\mathbf{\Phi}^+ = [\phi_1, \phi_2, \phi_3, \phi_4, ..., \phi_{p+1}] = [1, x, y, x^2, xy, y^2, ...]. \tag{4.17}$$

The approach to recovering the enriched solution is therefore to solve for $\boldsymbol{a}$. Finally, the anisotropy as well as the orientation of the frame field can be recovered using,

$$\rho = \left(\frac{A_1}{A_2}\right)^{\frac{-1}{p+1}} \tag{4.18}$$

$$\theta = \psi - \frac{\pi}{2} \tag{4.19}$$

Once the anisotropy and orientation of the frame field are identified, we can define the local continuous error estimator as follows,

$$e(\boldsymbol{x}, d, p) = B(\boldsymbol{x}, p)(d)^{-\frac{q(p+1)}{2}}, \tag{4.20}$$

where the term $B(\boldsymbol{x}, p)$ only depends on the anisotropy and orientation which was determined previously and is defined as follows,

$$B(\boldsymbol{x}, p) = 2^{\frac{q(p+1)+2}{2}} \left(\frac{2\pi}{q(p+1)+2} \right) (A_1(\boldsymbol{x}, p) A_2(\boldsymbol{x}, p))^{\frac{q}{2}}. \tag{4.21}$$

Now, the density function d remains to be solved in equation 4.20 globally through minimizing a global continuous error estimator that will be discussed in the next section.

4.2.2 Global Interpolation Error

The global continuous error estimator is derived by summing the local continuous error of the integration of the local continuous error estimator over the patchwise reconstructed domain, Ω_k which is composed of the current cell shown in orange, and four neighbor cells shown in green in Figure 4.1.

The global continuous error estimator is defined from [39] as,

$$\mathcal{E}(d, \mathcal{P}) = \sum_{k \in \mathcal{Q}_h} \int_{\Omega_k} e(\boldsymbol{x}_k, d_k, p_k) d\boldsymbol{x} = \sum_{k \in \mathcal{Q}_h} \| e^{int}_{\tilde{\boldsymbol{x}},p} \|^q_{L^q(\sum)} \geq \sum_{k \in \mathcal{Q}_h} \| e^{int}_{\tilde{\boldsymbol{x}},p} \|^q_{L^q(k)} \approx E_h, \tag{4.22}$$

where equation 4.22 provides an upper bound for the discrete minimization problem as stated in equation 3.8. Now, the remaining work is to minimize the global continuous error estimator in a similar fashion compared to the discrete variant of it while assuming uniform polynomial distribution, i.e. $\mathcal{P}(\boldsymbol{x}) = \mathcal{C}$, and this is defined as follows.

Given the existence of the exact solution $u \in C^{\infty}(\Omega)$ and a maximum number of degrees of freedom C, we seek a continuous mesh dependent on the density d and polynomial distribution $P(\boldsymbol{x})$ such that the global continuous error estimator $\mathcal{E}(d, \mathcal{P})$ is minimal,

$$\min_{d} \mathcal{E}(d, \mathcal{P}) = \int_{\Omega} B(\boldsymbol{x}, \mathcal{P}(\boldsymbol{x}))(d)^{-\frac{q(p+1)}{2}} d\boldsymbol{x} \tag{4.23}$$

$$\text{s.t.} \quad N_{hp}(d, \mathcal{P}) = \int_{\Omega} d(\boldsymbol{x})(\mathcal{P}(\boldsymbol{x}) + 1)^2 d\boldsymbol{x} \leq C$$

Now, to solve the minimization problem of the global continuous error estimator, we can employ a traditional constraint optimization framework such as through the use of Lagrange multipliers resulting in the following expression for the density,

$$d(\boldsymbol{x}) = \left(\frac{2\lambda(\mathcal{P}(\boldsymbol{x} + 1)}{qB(\boldsymbol{x}, \mathcal{P}(\boldsymbol{x}))} \right)^{\frac{2}{-q(\mathcal{P}(\boldsymbol{x})+1)+2}}, \quad \forall \boldsymbol{x} \in \Omega, \tag{4.24}$$

where the Lagrange multiplier λ can be found by solving the constraint of equation 4.23 through the Bisection method.

In summary, from the local error estimator, we can compute the orientation and the anisotropy which can be used to solve the minimization of the global continuous error. Then, once the minimization problem is solved, the density is found as well which completes the last part for constructing the frame field. The encoding of the frame field is done through the ".MSH" format where the information is encoded in the "Elements" section of the ".MSH" file. From there, the L_p-CVT algorithm reads this frame field map and performs mesh adaptation.

4.3 Goal-Oriented Approach

As discussed in section 3.3.1, a mesh can be adapted to reduce the error in a functional of interest by locally reducing the cell-wise DWR value which will globally reduce its error as shown in equation 3.28. Through the approach by Balan et al. [78], the idea is to refine regions in the mesh where the DWR is high and coarsen regions where the DWR value is low. To achieve this, the area of the cell will be of interest, and the area can be linked to the size of a frame field by the following,

$$h = \sqrt{\frac{1}{4}A(x)} = \sqrt{\frac{1}{4}\alpha_k I_k^c}, \tag{4.25}$$

where I_k^c is the current cell area, and α_k is a scaling factor that governs whether or not the cell should be refined or coarsened, and the target cell area is therefore,

$$I_k = \alpha_k I_k^c, \tag{4.26}$$

where the scaling factor α_k is a function of a logarithmic scaling factor ξ_k which is itself a function the cell-wise DWR value, η_k, which is defined as follows,

$$\alpha_k = \begin{cases} ((r_{\text{max}} - 1)\xi_k^2 + 1)^{-1}, & \eta_k \geq \eta_{\text{ref}}, \\ ((c_{\text{max}} - 1)\xi_k^2 + 1), & \eta_k < \eta_{\text{ref}}, \end{cases} \tag{4.27}$$

$$\xi_k = \begin{cases} \frac{\log(\eta_k) - \log(\eta_c)}{\log(\eta_{max}) - \log(\eta_c)}, & \eta_k \geq \eta_{\text{ref}}, \\ \frac{\log(\eta_k) - \log(\eta_c)}{\log(\eta_{min}) - \log(\eta_c)}, & \eta_k < \eta_{\text{ref}}, \end{cases} \tag{4.28}$$

where the maximum and minimum DWR values are defined as follows,

$$\eta_{\text{max}} = \max_{k \in \Omega_k} \eta_k, \tag{4.29}$$

$$\eta_{\text{min}} = \min_{k \in \Omega_k} \eta_k. \tag{4.30}$$

Now, three parameters of choice as defined by the user are respectively r_{max} (maximum refinement factor), c_{max} (maximum coarsening factor), and η_{ref} (a reference DWR value). If the DWR value η_k is greater than or equal to the reference DWR value, then the cell is refined, otherwise, the cell is coarsened. Now, to ensure that the output mesh respects the target complexity C, we use the bisection method to find the η_{ref} which results in a target complexity of C with the selected parameter of choices. We first evaluate the α_k parameter, then we define the target size h which we will use to compute the complexity which is evaluated using equation 4.4, then, the functional of choice for the bisection method would be the current complexity minus the target complexity, and therefore, the algorithm would converge once the current and target complexities are equalized.

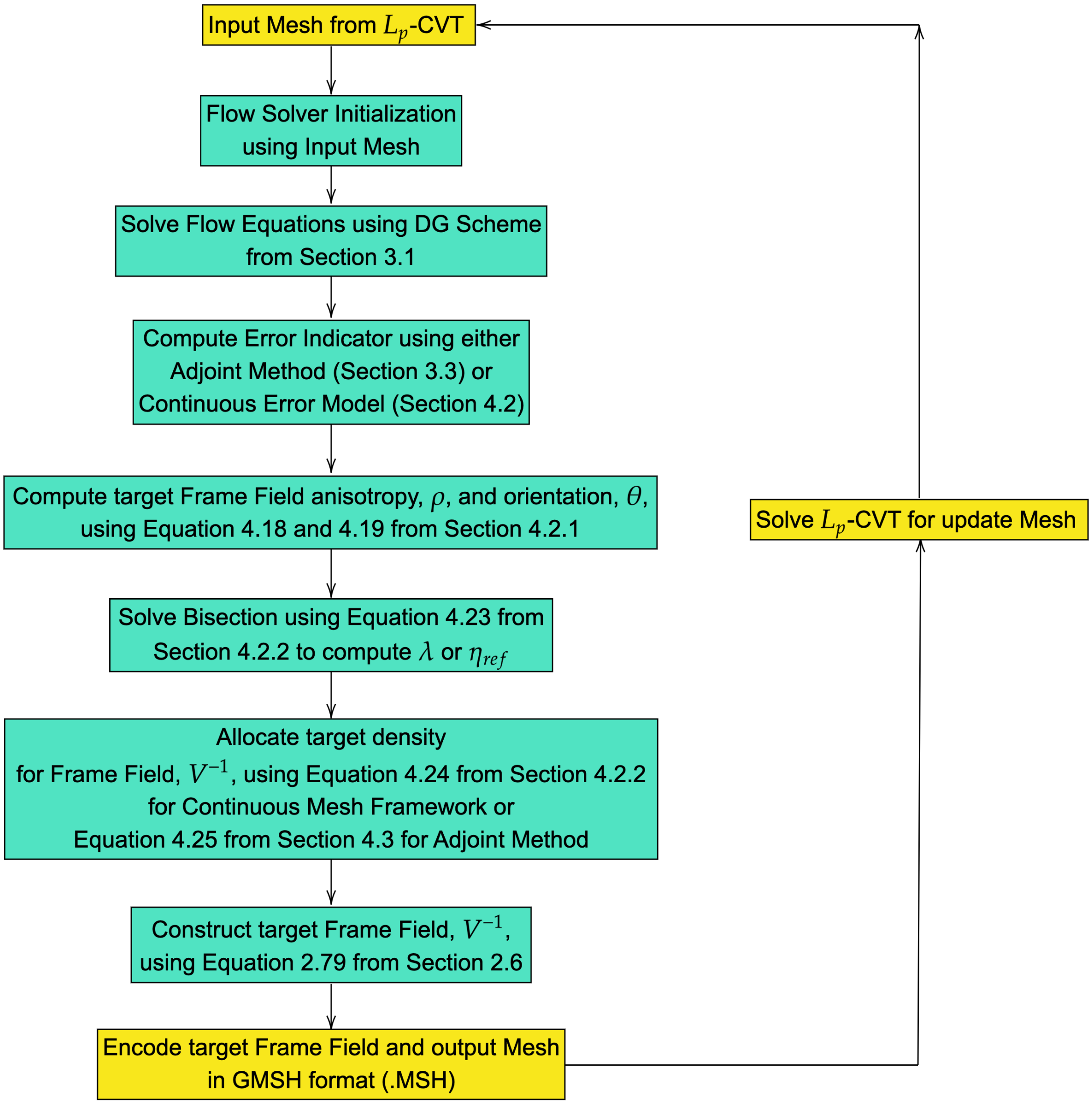

Input Mesh from L_p-CVT
Flow Solver Initialization using Input Mesh
Solve Flow Equations using DG Scheme from Section 3.1
Compute Error Indicator using either Adjoint Method (Section 3.3) or Continuous Error Model (Section 4.2)
Compute target Frame Field anisotropy, ρ, and orientation, θ, using Equation 4.18 and 4.19 from Section 4.2.1
Solve Bisection using Equation 4.23 from Section 4.2.2 to compute λ or η_{ref}
Allocate target density for Frame Field, V^{-1}, using Equation 4.24 from Section 4.2.2 for Continuous Mesh Framework or Equation 4.25 from Section 4.3 for Adjoint Method
Construct target Frame Field, V^{-1}, using Equation 2.79 from Section 2.6
Encode target Frame Field and output Mesh in GMSH format (.MSH)
Solve L_p-CVT for update Mesh

Figure 4.2: Flowchart showing the flow solver coupled with the L_p-CVT algorithm

Chapter 5

Results

In this section, a series of test cases will be presented to confirm the applicability of the boundary treatment for anisotropic all-quad mesh adaptation through an L_p-CVT approach. The goal is to demonstrate the applicability of the boundary treatment with analytical metric fields as well as discrete metric fields and how it compares with the performance of the same test case without boundary treatment. The benefits as well as the drawbacks will be discussed. Then, a flow solver test case simulating a transonic flow over a NACA0012 will be presented with the boundary treatment method and this will showcase the application of this technique to a practical engineering scenario.

5.1 Analytical Metric Field

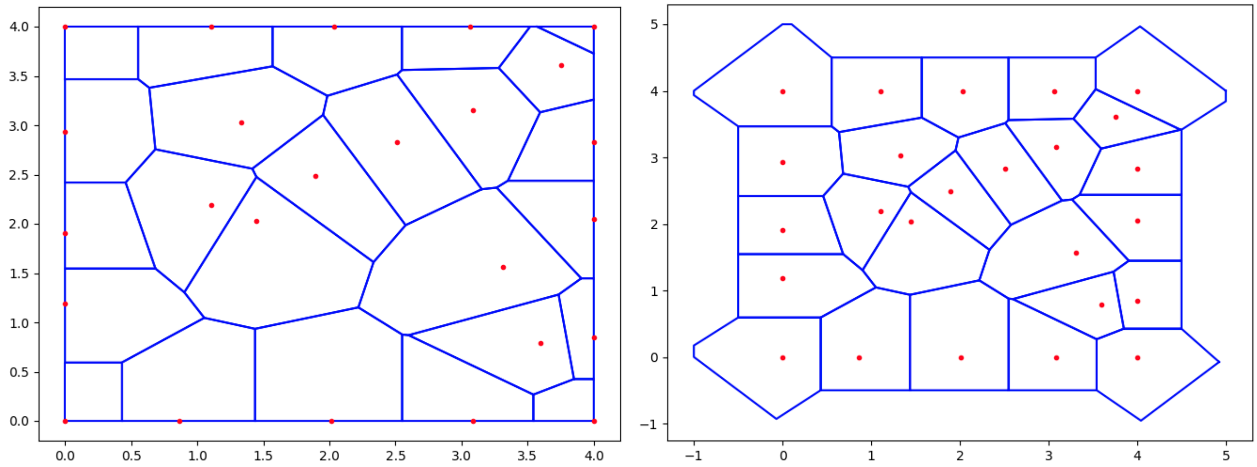
The analytical metric field is a continuous metric field that is defined at every position $\boldsymbol{x}$ in the domain Ω. These are different compared to a discrete metric field as an analytical metric field does not depend on a background metric field which essentially defines discretely a metric field for every cell in the mesh. In contrast, each point $\boldsymbol{x}$ in the domain is exactly defined by a metric function (i.e. there are variations in the metric field defined within the cell) for the analytical case. We will first examine how the boundary treatment method, as defined in section 2.11, works on simple analytical metric fields such as a constant metric field, then, we will also investigate more challenging analytical metric fields such as a quadratic metric field. Note that as mentioned before in section 2.6, for all test cases presented here, an L_p norm of six will be employed since values exceeding six do not provide any additional benefit in estimating the L_∞ norm. Additionally, unless otherwise specified, the input Voronoi diagram for the static boundary cases conforms to the underlying metric field on the principle of section 2.11 to allow for convergence.

5.1.1 Constant Metric Field

The first analytical metric field that is of interest is the constant metric field defined as follows,

$$\mathcal{M} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.1}$$

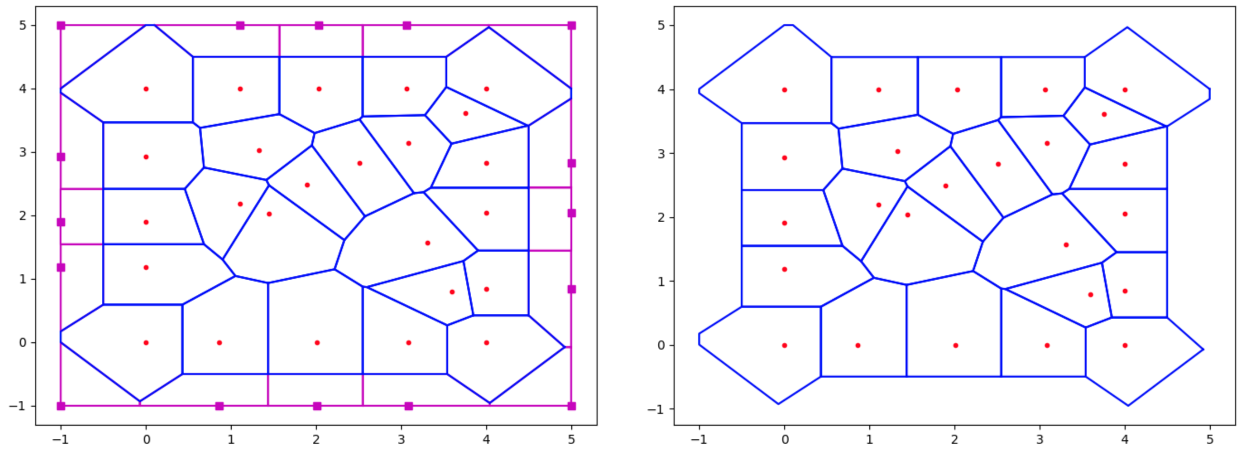
The constant metric field is the simplest form of a metric field that can exist and does not vary in space, therefore, it acts as the baseline and benchmark case to prove the validity of the boundary treatment method applied to the L_p-CVT algorithm.



(a) Clipped Boundary Voronoi Diagram **(b)** Reconstructed Boundary Voronoi Diagram

Figure 5.1: Comparison of clipped and reconstructed boundary Voronoi diagram based on a constant metric field

The first validity check that is of interest is the boundary reconstruction technique provided in section 2.9. Figure 5.1 shows an example of the reconstructed boundary shown on the right-hand side. Note that the boundary cells are reconstructed based on a set of ghost cells and this is shown in Figure 5.2 where the ghost cells or the ghost boundary is highlighted in magenta.



(a) Reconstructed Voronoi Diagram with Ghost Cells

(b) Reconstructed Voronoi Diagram

Figure 5.2: Reconstructed boundary with ghost cells

The gradient of the energy computed at the boundary (for both the bottom and right-sided boundary nodes) is summarized in the following table,

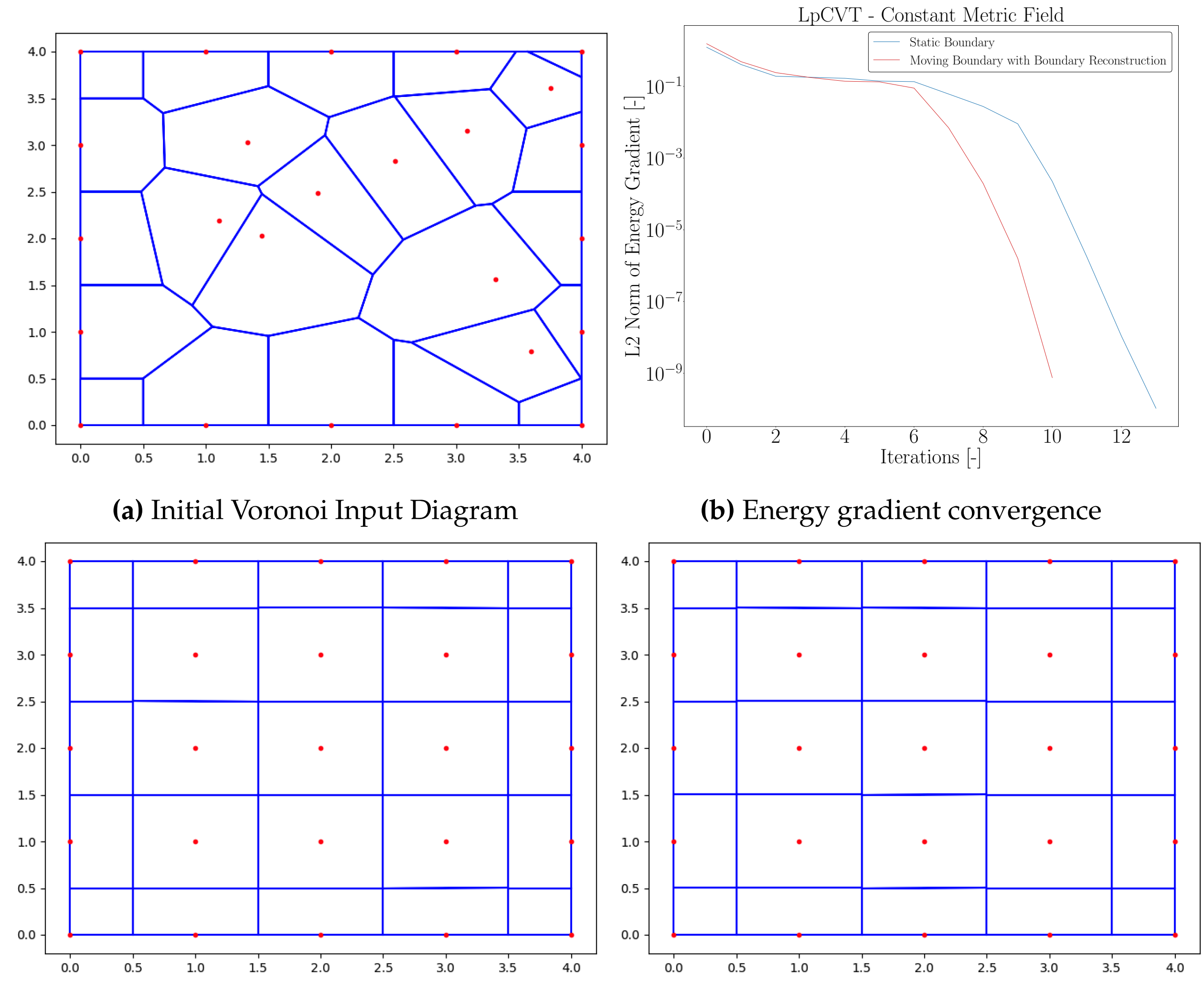
x_i	1.0	2.0	3.0	4.0	4.0	4.0
y_i	0.0	0.0	0.0	1.0	2.0	3.0
$\frac{dE}{dx_i}_C$	-4.10885e-13	-3.01231e-13	-2.33456e-13	0.0396205	0.0396205	0.0396205
$\frac{dE}{dy_i}_C$	-0.0396205	-0.0396205	-0.0396205	-7.18901e-13	-7.80369e-13	8.2186e-14
$\frac{dE}{dx_i}_R$	-1.27378e-10	2.11382e-10	1.64414e-10	-1.28909e-10	3.9692e-11	6.08676e-11
$\frac{dE}{dy_i}_R$	-2.76333e-10	-1.07701e-10	1.38448e-10	-7.7596e-11	-1.48943e-10	5.26289e-11

Table 5.1: Energy gradient value for clipped boundary ($\frac{dE}{dx_i}_C$) and reconstructed boundary ($\frac{dE}{dx_i}_R$) at the bottom and right side boundary

Table 5.1 shows the energy gradient for the bottom side as well as the right side of the boundary. For the bottom side (i.e. points at y_i equals 0), the clipped boundary energy gradient $\frac{dE}{dx_i}_C$ shows convergence on the x-direction with tolerance up to $\mathcal{O}(1\text{e-}13)$, however, the y-direction gradient only converges up to $\mathcal{O}(1\text{e-}2)$. This drastic difference in convergence can be explained by the fact that the boundary node energy gradients are evaluated with a clipped Voronoi cell. This limitation is due to the boundary being predefined by a rectangular boundary and hence, the full Voronoi cell cannot be used for the evaluation of the gradient. For the energy value to be at an optimal state, the boundary points on the bottom side must further move in the y-direction, which is physically not possible on the boundary. Hence, the evaluation of the gradient with a clipped equation limits the optimal state found with the LBFGS algorithm as the boundary energy gradients do not fully reflect the actual full cell at the boundary. Additionally, at convergence, all three y-direction clipped energy gradients are identical, showing that the constant metric field is being well applied in the code requiring the three boundary nodes to move in the same direction and by the same amount.

In contrast, the reconstructed boundary cell shows convergence in both the x and y-directions which is much better compared to the clipped version. The convergence as shown in Table 5.1 shows convergence up to $\mathcal{O}(1\text{e-}10)$ and $\mathcal{O}(1\text{e-}11)$ where the y-direction energy gradient is fully recovered and is of the same magnitude as the x-direction. Note that the tolerance is not as low as the clipped boundary case (about one to two orders higher), however, the reconstructed boundary case does show a tolerance that is deemed acceptable for determining a convergence. This may be because a simplified version of the boundary reconstruction method is used (as demonstrated in section 2.9) instead of the cell-wise boundary reconstruction method used by Ekelschot [68]. For our work, the simplified version was implemented instead due to the structure of our code, but, the drawbacks are not apparent, and in contrast, acceptable convergence is shown, thus, the simplified version of the boundary reconstruction will be used in the following section with the moving boundary method.

For further validation, the right-side energy gradient is also investigated and similar results are observed. Since the right-side boundary nodes are constrained in the y-direction, the energy gradient convergence is therefore achieved by the y-direction (i.e. the boundary points no longer move in the y-direction, and have reached an optimal state). On the other hand, the x-direction energy gradient shows convergence up to $\mathcal{O}(1\text{e-}2)$, similar to the bottom side. With the reconstructed boundary, the energy gradient is again recovered where both the x and y-direction energy gradients converge at the same magnitude.



(a) Initial Voronoi Input Diagram

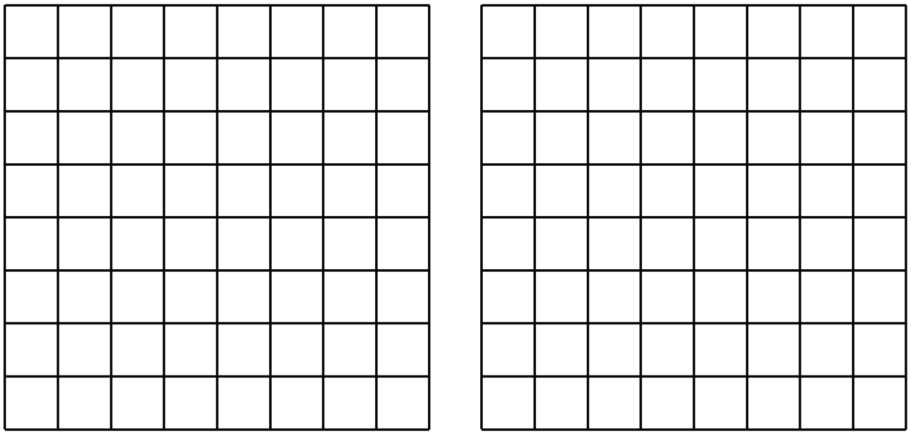
(b) Energy gradient convergence

(c) Static Boundary Output Voronoi diagram

(d) Moving Boundary Output Voronoi diagram

Figure 5.3: Constant Metric Field Result Summary

For the subsequent analysis in this section, the reconstructed boundary method will be utilized for evaluating the boundary energy gradient. Figure 5.3 shows the result of the L_p-CVT algorithm solved with both the static and moving boundary method for the constant metric field. The test case is set up with an initial number of 9 randomly distributed volume points and 16 boundary points for a total of 25 points in a domain $\Omega = [0, 4]^2$. Note that no edge splitting is utilized in this test case as only the first energy gradient convergence is of interest for a proof of concept. Unsurprisingly, both the static and moving boundary method converges to the same output plot as shown in Figure 5.3. Additionally, the energy gradient convergence is very similar (up to $\mathcal{O}(1\text{e-}9)$) with the moving boundary showing a steeper decrease in energy gradient at around the 6th iteration compared to the static boundary which decreases slower and requires two additional iterations for convergence. Figure 5.4 shows the all-quad mesh output for both the static and moving boundary approach, and both output meshes are identical.

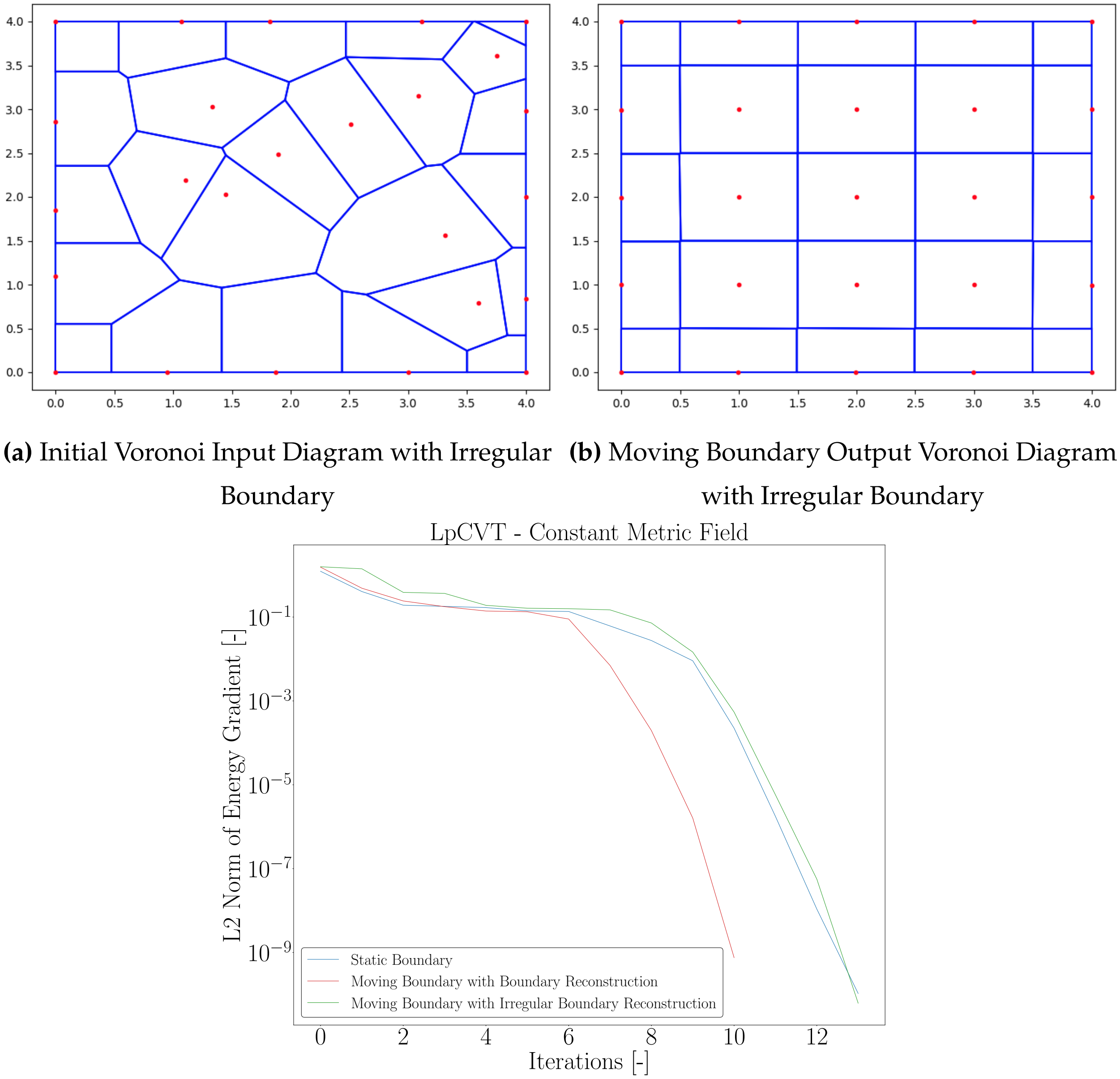


(a) Static Boundary All-Quad GMSH Output

(b) Moving Boundary All-Quad GMSH Output

Figure 5.4: Constant Metric Field All-Quad Mesh Summary

Another interesting test case was performed for the moving boundary approach which is to begin the optimization with an initial Voronoi diagram with irregular boundaries. This implies that the boundary points are distorted compared to the regular boundary which was used in the previous results. This test case shows the strength of the moving boundary approach where the starting boundary distribution can be decoupled from the underlying metric field. This method does not necessarily require the boundary to follow the metric field for it to converge. The output mesh is similar to the regular boundary case as shown in Figure 5.5, and the convergence of the gradient is also promising where it matches closely the static boundary method with the regular boundary.



(a) Initial Voronoi Input Diagram with Irregular Boundary

(b) Moving Boundary Output Voronoi Diagram with Irregular Boundary

(c) Energy gradient convergence

Figure 5.5: Constant Metric Field Result with Irregular Boundary

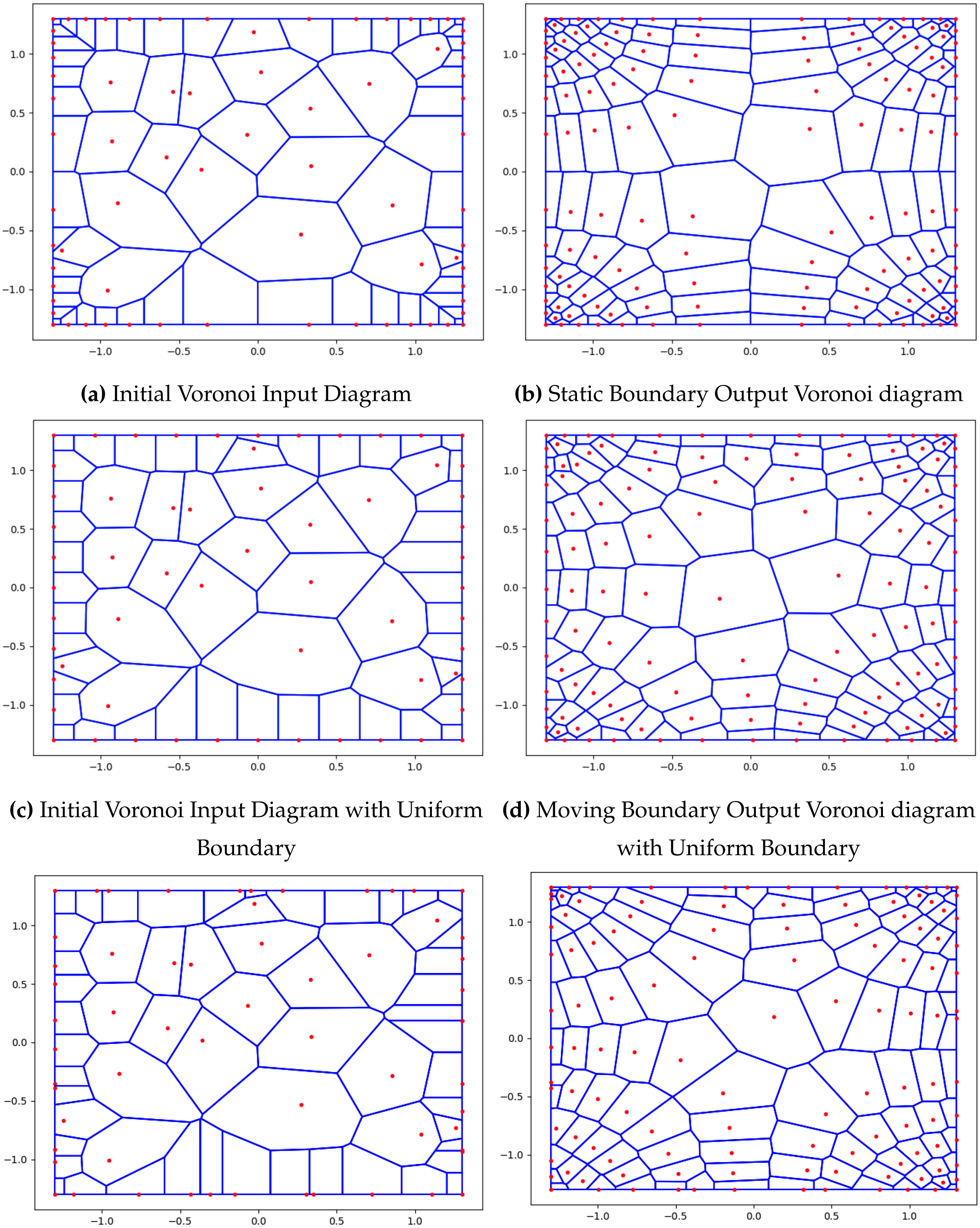
5.1.2 Quadratic Metric Field

The second analytical metric field that is of interest is the quadratic metric field defined as follows,

$$\mathcal{M}(x,y) = \begin{bmatrix} 1+4x^2 & -4xy \\ -4xy & 1+4y^2 \end{bmatrix} \tag{5.2}$$

Notice here that this metric field varies with respect to the x and y-coordinate in the domain Ω and it has been explored in previous work [18, 39, 68]. The test case is set up with an initial number of 20 randomly distributed volume points and 40 boundary points for a total of 60 points in the domain $\Omega = [-1.3, 1.3]^2$. For the moving boundary, the initial Voronoi diagram is set up with uniform and irregular boundaries to demonstrate the capability of the moving boundary method. To achieve a quasi-unit mesh, the edge splitting and merging threshold are set to $L_{\mathbf{M}} \in [0.0, 2.0]$ for static boundary and $L_{\mathbf{M}} \in [0.005, 2.0]$ for moving boundary. In contrast to the work of Maclean in [39] where the author does not introduce any merging thresholds for the static boundary, the moving boundary approach requires at minimum a small merging threshold to avoid overlapping of boundary points which can create extra edges if not removed/merged.

Figure 5.6 shows the results of the static and moving boundary approach. The static boundary approach shows a boundary conforming to the quadratic metric field which is symmetric on each side. As a result, the final output mesh is almost symmetric along the diagonal axis. For the uniform and irregular moving boundary approach, the resulting boundary distribution upon convergence reassembles a lot of the initial uniform and distorted boundary which suggests a solution that cannot be achieved using a static boundary. Additionally, it is apparent that near the four corners, there are clustering of points resulting from the quadratic metric field, however, the clustering effect is not as strong as the static boundary since more boundary points moved toward the center of the domain near $x = 0$ and $y = 0$.



(a) Initial Voronoi Input Diagram

(b) Static Boundary Output Voronoi diagram

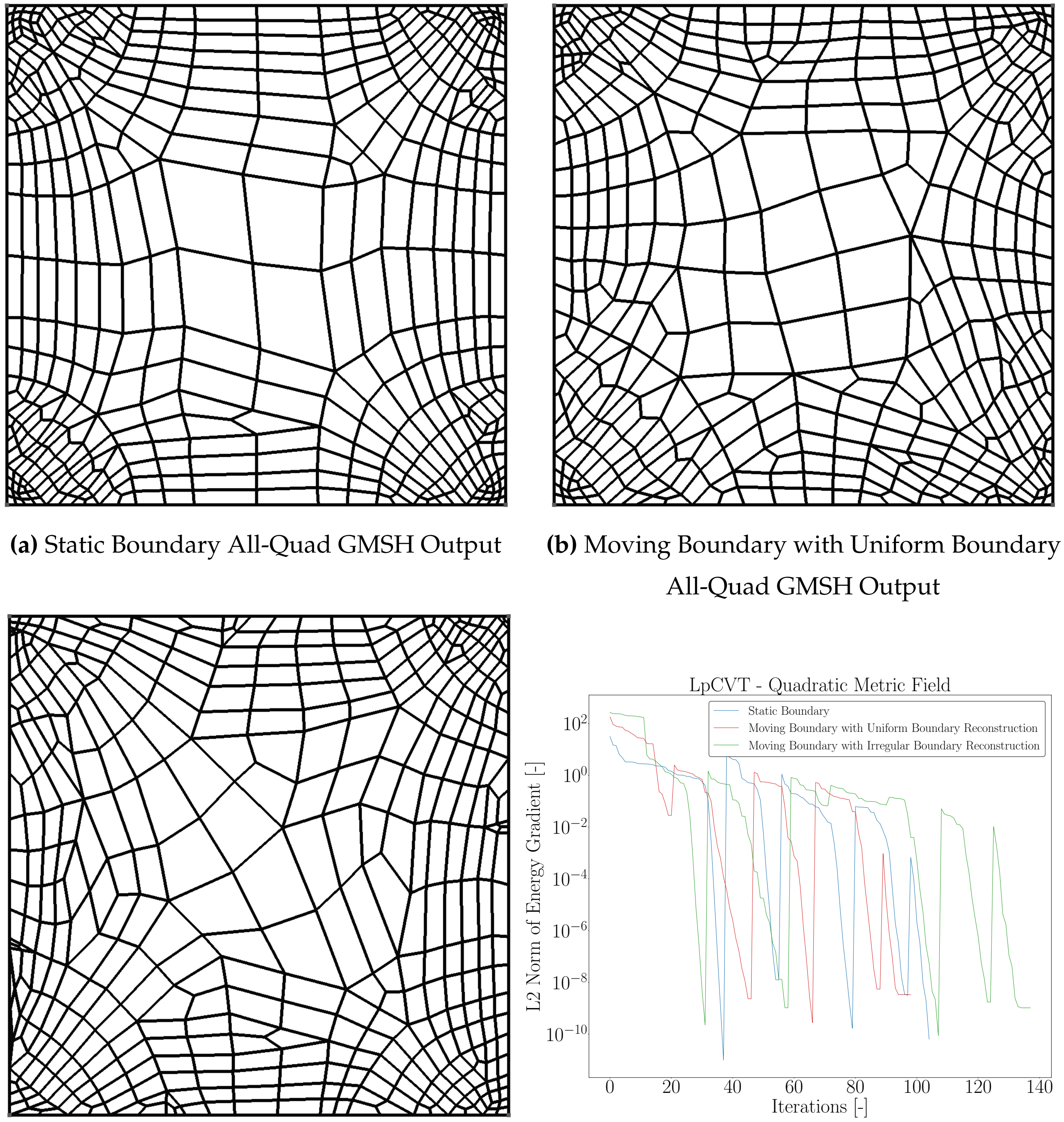
(c) Initial Voronoi Input Diagram with Uniform Boundary

(d) Moving Boundary Output Voronoi diagram with Uniform Boundary

(e) Initial Voronoi Input Diagram with Irregular Boundary

(f) Moving Boundary Output Voronoi diagram with Irregular Boundary

Figure 5.6: Quadratic Metric Field Result Summary



(a) Static Boundary All-Quad GMSH Output

(b) Moving Boundary with Uniform Boundary All-Quad GMSH Output

(c) Moving Boundary with Irregular Boundary All-Quad GMSH Output

(d) Energy gradient convergence

Figure 5.7: All-Quad Output Mesh and Energy Gradient

Figure 5.7 shows the all-quad mesh output from GMSH for each approach as well as the energy gradient convergence. One major observation is that the all-quad mesh output is much more structured for the static boundary approach compared to the moving boundary approach. This is not surprising since the boundary nodes are more constrained throughout the optimization phase with a symmetric boundary distribution which adds order to the cells upon convergence. On the other hand, the moving boundary approach provides additional flexibility to the algorithm to place boundary points at locations of interest, which results in less patterned output mesh configurations driven by the movement of boundary nodes. Nevertheless, allowing the merging of points to occur with the moving boundary approach, fewer points are required in the output mesh which results in fewer cells generated. The convergence data is summarized in the following table,

Boundary Condition	**#Nodes**	**#Quadrilaterals**	**Iteration**	$\|\nabla E\|_2$
Static Boundary	513	460	104	6.36196e-11
Moving Uniform Boundary	481	432	98	3.3297e-09
Moving Irregular Boundary	463	416	137	1.03667e-09

Table 5.2: Summary of convergence for Quadratic Metric All-Quad Mesh and $\|\nabla E\|_2$

In terms of energy gradient convergence, all three approaches appear to behave very similarly with the uniform as well as the irregular moving boundary approach showing a plateau near the 94^{th} and 135^{th} iteration respectively which is due to the convergence on the step size tolerance. The static boundary approach converges to $\mathcal{O}(\text{1e-11})$ whereas the moving boundary approach converges to $\mathcal{O}(\text{1e-9})$. The difference of about two orders of convergence can be explained by the fact that the simplified boundary reconstruction scheme is used instead of the cell-wise boundary reconstruction method used by Ekelschot [68]. A simpler scheme is used to achieve expected convergence at the cost of reduced accuracy on the gradient evaluation which is deemed acceptable for this work. Nevertheless, the behavior of spike down and up in the energy gradient is a result of the split and merge algorithm acting on the outer loop of the LBFGS scheme which essentially restarts the optimization process after each convergence until only 2% of the total number of edges is required for split or merge where the algorithm stops.

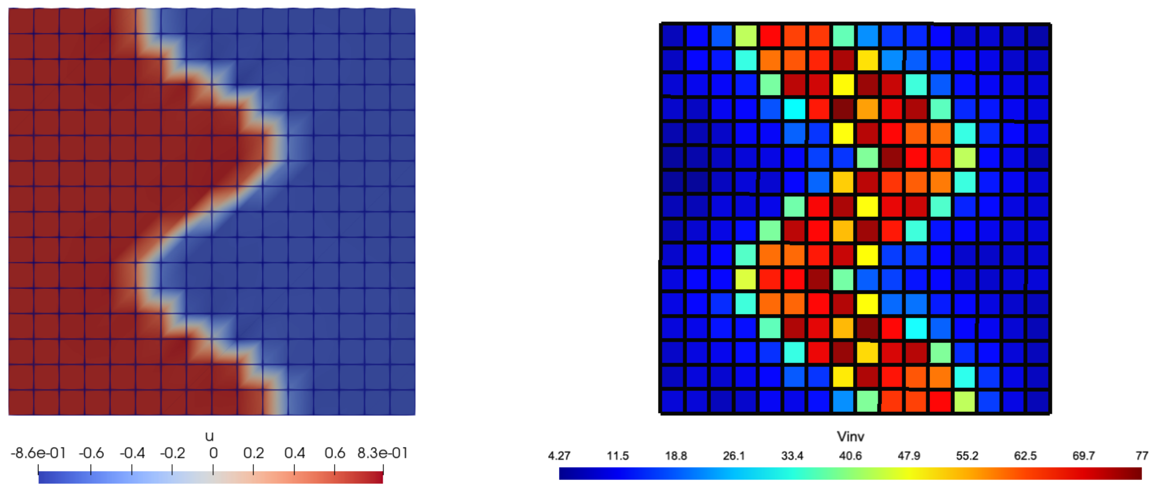
5.2 Discrete Frame Field

Unlike analytical metric fields, discrete frame fields are not continuous across the domain Ω. In contrast, each cell is discretely defined by a frame field as explained in Chapter 2, and the frame field is generated using the continuous mesh model as explained in Chapter 4. This section aims to explore the applicability of the moving boundary approach on discrete frame fields, which will serve as a basis for the flow solver test case that will be discussed shortly. Note that the encoded frame field is defined as V^{-1} on each cell.

5.2.1 S-Shock Adaptation

The s-shock test case is an advection-diffusion problem with a manufactured solution S. For this test case, an initial p_1 solution on a 16 x 16 grid is solved and a feature-based error estimator will be used for encoding the frame fields. The governing equation is defined as follows,

$$\nabla \cdot (D\nabla u) - C \cdot \nabla u = S(u), \tag{5.3}$$



(a) Initial S-Shock solution $(p = 1)$

(b) Initial S-Shock Frame Field Map

Figure 5.8: Feature-based Initial S-Shock Solution

where,

$$D = \frac{0.01\pi}{e}\begin{bmatrix}12 & 3\\ 3 & 20\end{bmatrix}, \quad C = \begin{bmatrix}1.1, & -\frac{\pi}{e}\end{bmatrix}^T, \tag{5.4}$$

and the manufactured solution is given by the function $S(u)$,

$$u(x,y) = 0.75\tanh\big(4\sin(10y-5) - 24x + 12\big). \tag{5.5}$$

Figure 5.8 shows the initial solution as well as the initial frame field output with the L_2 norm of the V^{-1} frame field shown on the right which will be used as an input to the L_p-CVT algorithm. First, the boundary reconstruction approach with moving boundary is studied on the initial solution. Table 5.2 shows that reconstructing the boundary using the simple approach as opposed to the cell-wise method from [68] does not provide significant benefit to the energy gradient convergence. Most notably, the evaluation of the reconstructed boundary length resulting in $\|\boldsymbol{u}\|_f = 1$ can be quite different from boundary to boundary at the start of the mesh adaptation cycle. This is because no frame field interpolation or extrapolation is performed, and only association of the frame field nearest to the boundary points is used, therefore, accurate evaluation of the gradient may be unachievable. Additionally, in the simple boundary reconstruction approach, outliers can overweight the entire boundary reconstructed length l_r (i.e., if one boundary frame field requires a reconstructed boundary length $l_{b,i}$ of 60, whereas the other boundary points only require a reconstructed boundary length of 0.2 to result in $\|\boldsymbol{u}\|_f = 1$, then the final reconstructed boundary length using the simple approach may be drastically higher than 0.2, which impacts the accurate evaluation of the energy gradient). Three grey rows are highlighted in Table 5.2 to demonstrate outliers which drives the l_r value considerably higher. A single light cyan row is also highlighted to show a higher $\|\boldsymbol{E}\|_2$ value compared to the other boundary points at convergence. Therefore, the simple approach for reconstructing the boundary does not show satisfactory results for discrete frame fields and will not be used in the analysis of discrete frame fields.

#Node	x_i	y_i	$l_{b,i}$	$l_r = \frac{\sum_i^{n_b}(l_{b,i})\|_{l_{\mathcal{M}}=1}}{n_b}$	$l_r/l_{b,i}$	$\|\nabla \boldsymbol{E}\|_2$
1	0.111111	0	0.175599	3.957021	22.53441648	0.00188491
2	0.222222	0	0.654876	3.957021	6.042397339	0.0226864
3	0.333333	0	0.81468	3.957021	4.857147592	0.00849525
4	0.444444	0	5.92369	3.957021	0.667999338	0.269867
5	0.555556	0	0.291196	3.957021	13.58885768	0.937548
6	0.666667	0	0.17281	3.957021	22.89810196	0.462133
7	0.777778	0	0.247215	3.957021	16.00639524	0.753397
8	0.888889	0	18.9879	3.957021	0.208396979	0.0102965
9	1	0.111111	0.369684	3.957021	10.70379297	0.00743074
10	1	0.222222	0.147742	3.957021	26.78331822	0.0252558
11	1	0.333333	0.128109	3.957021	30.88792357	0.000198845
12	1	0.444444	0.12143	3.957021	32.58684839	0.00063762
13	1	0.555556	0.106829	3.957021	37.04070056	0.000464526
14	1	0.666667	0.104173	3.957021	37.98509211	0.04506
15	1	0.777778	0.103097	3.957021	38.3815339	0.626763
16	1	0.888889	0.326615	3.957021	12.11524578	0.00920862
17	0.888889	1	3.30233	3.957021	1.198251235	0.00146308
18	0.777778	1	1.83148	3.957021	2.16055922	0.000079735
19	0.666667	1	1.12603	3.957021	3.514134615	0.0154094
20	0.555556	1	1.26035	3.957021	3.13962074	0.000378001
21	0.444444	1	0.209269	3.957021	18.9087777	0.819118
22	0.333333	1	0.170561	3.957021	23.20003401	8.67232
23	0.222222	1	0.27236	3.957021	14.52864224	0.0221344
24	0.111111	1	60.2364	3.957021	0.065691525	0.732611
25	0	0.888889	1.8411	3.957021	2.149270002	0.0045626

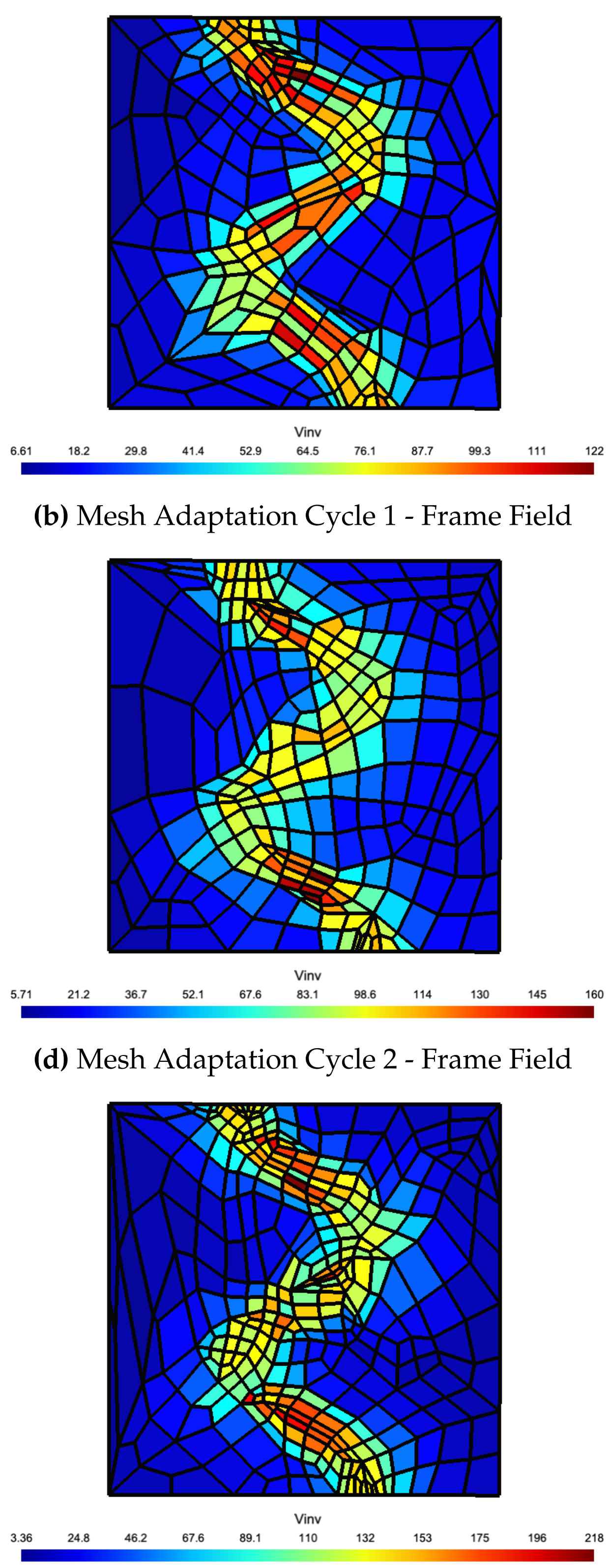
Table 5.3: Summary of reconstructed length for 25 boundary points based on the nearest frame field of the initial S-Shock solution

The test case is set up with an initial number of 70 randomly distributed volume points and an arbitrary number of boundary points as defined by section 2.10 in a domain $\Omega = [0, 1]^2$. Figure 5.9 shows the results of the L_p-CVT algorithm after using the moving boundary approach and Figure 5.10 shows the results of static boundary for a total of five mesh adaptation iterations. The L_p-CVT algorithm is performed using split and merge thresholds of $L_{\mathbf{M}} \in [0.25, 2.25]$, ensuring a quasi-unit mesh with a considerable amount of tolerances on the target edge sizes. Clearly, the target output mesh of the fifth adaptation cycle shows the best overall alignment of the mesh with the s-shock for both approaches.

Noticeably, the moving boundary approach shows a greater amount of boundary mesh points compared to the static boundary approach at each mesh adaptation cycle. This is expected as the moving boundary approach does not fix the number of boundary points in contrast to the static boundary approach and allows for the points to move. As a result, a more structured quad-dominant mesh near the boundary (far from the s-shock) emerges compared to the static boundary case where the distribution of the points at the boundary is predefined. This comparison is shown in Figure 5.13 a) and b).

Additionally, Figure 5.10 e) and Figure 5.12 e) show the convergence of the L_p-CVT energy gradient for both approaches. As expected, for a discrete frame field, since the cell-wise frame field is only a discrete representation of the actual frame field, the L_p-CVT energy gradient shows a region of stalling (i.e. flat lines) where the energy gradient has difficulty converging. This same phenomenon has been observed in [39] when the output adapted mesh based on a discrete as well as an analytical frame field were compared. However, the moving boundary approach does show much more difficulty in achieving a lower order of convergence compared to the static boundary approach. One possible reason behind this is that the boundary gradients are not fully resolved with local cell boundary reconstruction, ultimately stalling the convergence. In addition, the moving boundary approach shows a smaller drop in order of convergence at the start of the mesh adaptation cycle (i.e. drop of about 1 to 2 orders of convergence). However, as more and more cells are being adapted towards regions of large directional changes in the solution, the frame field becomes much more refined, resulting in much better convergence at the fifth mesh adaptation cycle (i.e. drop of about 3 to 4 orders of convergence). The same is true for the static boundary approach, however, since only volume nodes are of interest, the energy gradient convergence is a much better order of convergence decrease compared to the moving boundary approach.

In terms of the L_2 norm of the solution error, Figure 5.13 d) shows that a uniform grid performs the worst when refining the mesh uniformly, while the moving and static boundary approaches show similar performance with an almost equal number of degrees of freedom (DOFs). Note that during the early mesh adaptation cycles, both the static and moving boundary approaches have divergences. The first mesh adaptation cycle for the static boundary approach converges to a lower DOFs mesh compared to the baseline mesh. This happens due to the algorithm converging with respect to the step length of the LBFGS algorithm, therefore, the output mesh is considered as an optimal mesh since no further solutions can be found in the proposed search direction. This then creates a small red line pointing towards 930 DOFs in Figure 5.13 and goes back to 1030 DOFs on the second mesh adaptation cycle. Similarly, the moving boundary approach also experiences the same converging scenario an increase in the L2 norm of the solution error is observed at the second mesh adaptation cycle. Besides, the static boundary approach shows a stalling in the convergence of the error after around 2000 DOFs, whereas the moving boundary approach still shows a downward slope in the error convergence plot, suggesting that this method can achieve much lower error convergence due to the benefit of allowing boundary points to move. Lastly, both the static and moving boundary approaches show a similar progression in the number of DOFs increase per mesh adaptation cycle since the target complexity grows by 1.5 times per cycle starting with an initial target complexity of 1536.



(a) Mesh Adaptation Cycle 1 - Solution

(b) Mesh Adaptation Cycle 1 - Frame Field

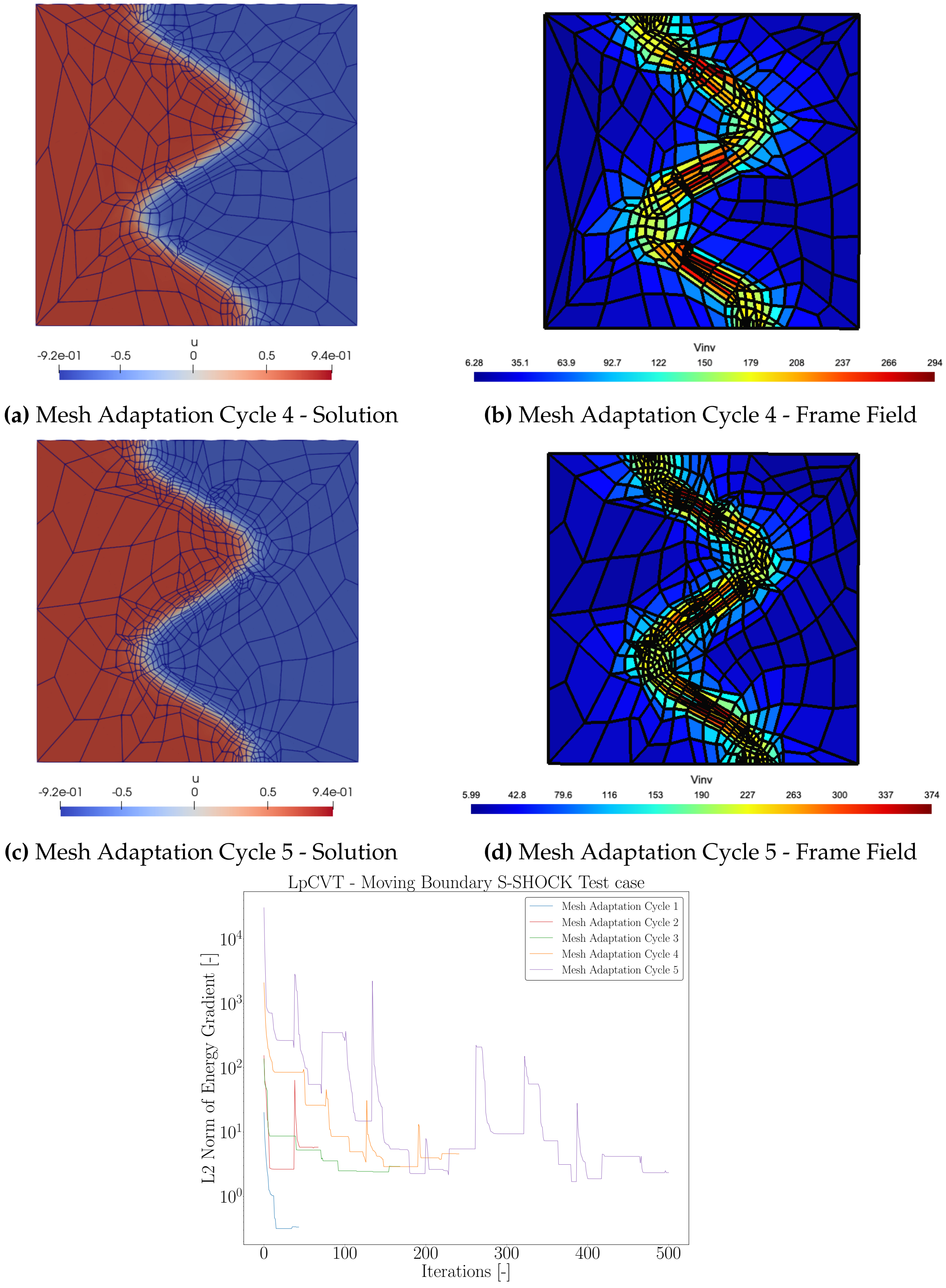
(c) Mesh Adaptation Cycle 2 - Solution

(d) Mesh Adaptation Cycle 2 - Frame Field

(e) Mesh Adaptation Cycle 3 - Solution

(f) Mesh Adaptation Cycle 3 - Frame Field

Figure 5.9: Summary of S-Shock Case Results Set 1 - Moving Boundary



(a) Mesh Adaptation Cycle 4 - Solution

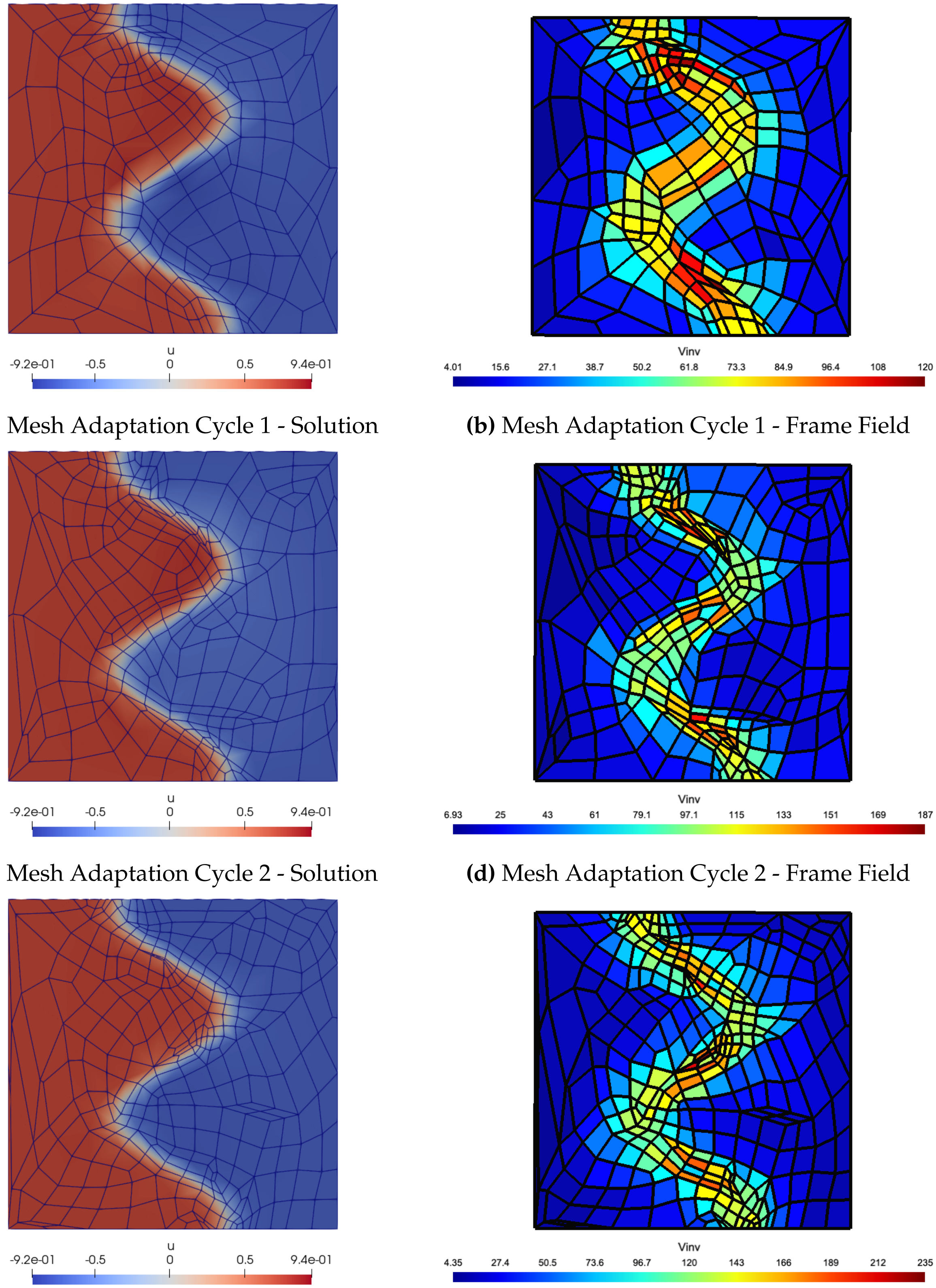
(b) Mesh Adaptation Cycle 4 - Frame Field

(c) Mesh Adaptation Cycle 5 - Solution

(d) Mesh Adaptation Cycle 5 - Frame Field

(e) L_p-CVT - L_2 norm Energy Gradient

Figure 5.10: Summary of S-Shock Case Results set 2 - Moving Boundary



(a) Mesh Adaptation Cycle 1 - Solution

(b) Mesh Adaptation Cycle 1 - Frame Field

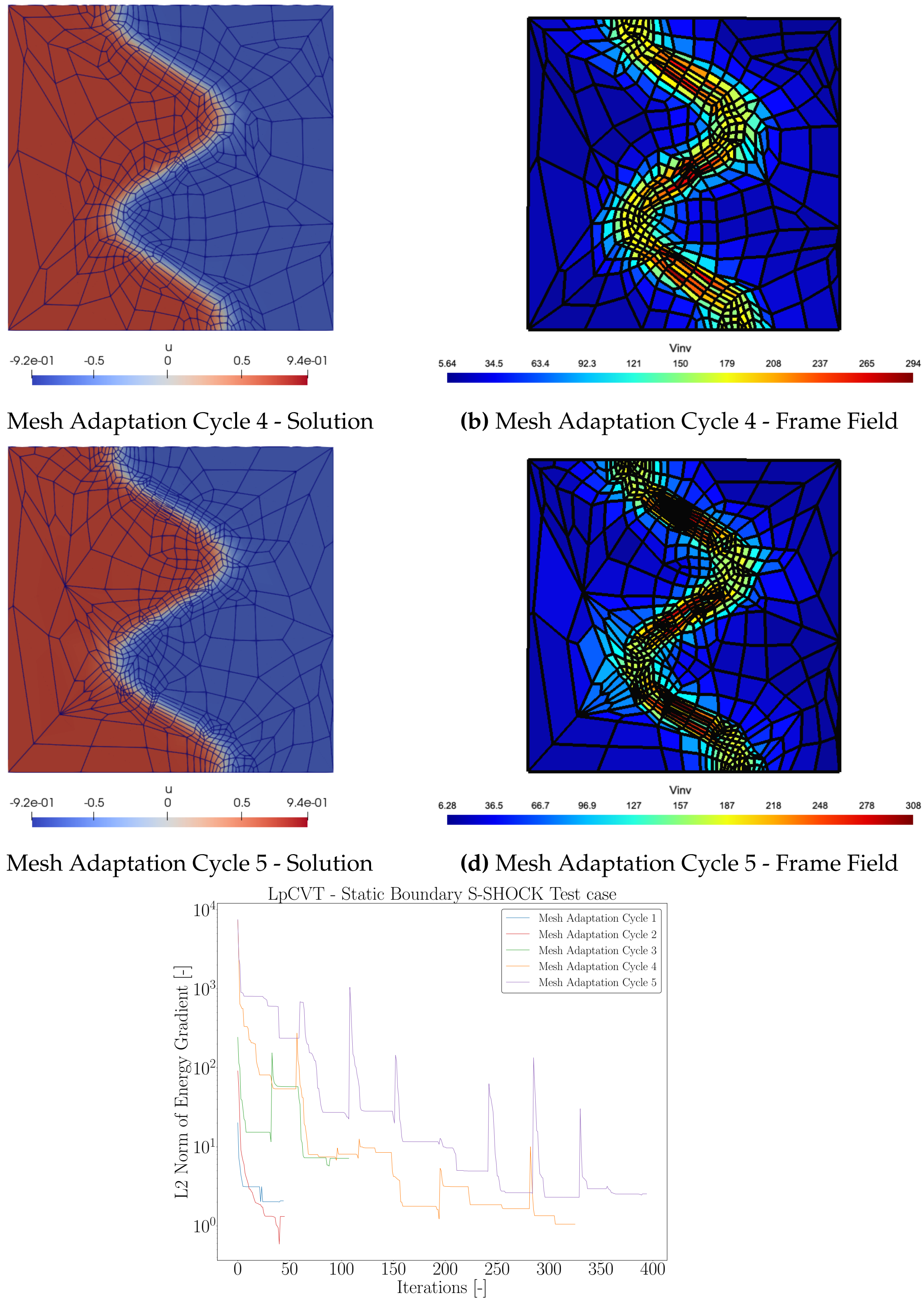
(c) Mesh Adaptation Cycle 2 - Solution

(d) Mesh Adaptation Cycle 2 - Frame Field

(e) Mesh Adaptation Cycle 3 - Solution

(f) Mesh Adaptation Cycle 3 - Frame Field

Figure 5.11: Summary of S-Shock Case Results Set 1 - Static Boundary



(a) Mesh Adaptation Cycle 4 - Solution

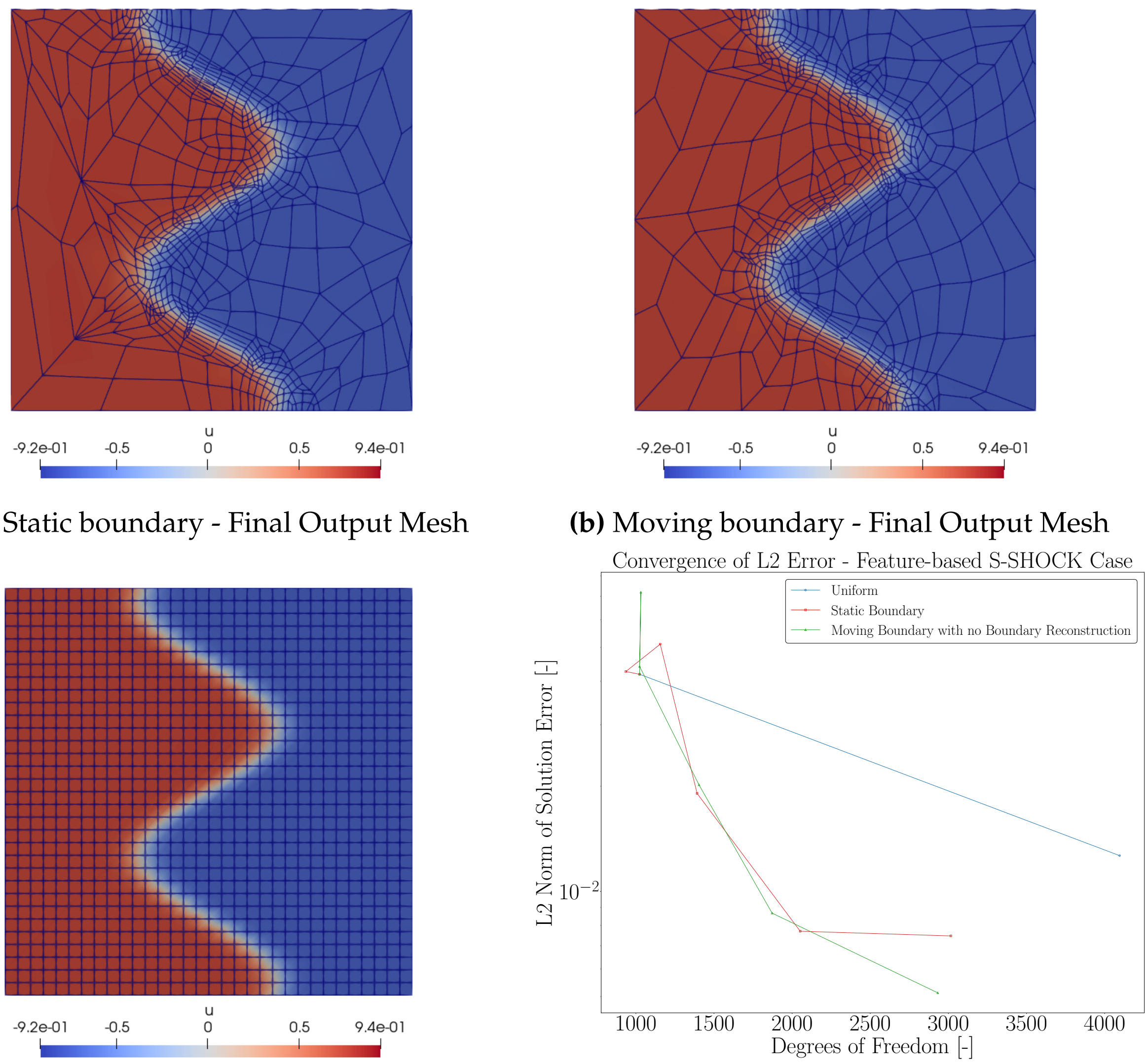
(b) Mesh Adaptation Cycle 4 - Frame Field

(c) Mesh Adaptation Cycle 5 - Solution

(d) Mesh Adaptation Cycle 5 - Frame Field

(e) L_p-CVT - L_2 norm Energy Gradient

Figure 5.12: Summary of S-Shock Case Results Set 2 - Static Boundary



(a) Static boundary - Final Output Mesh

(b) Moving boundary - Final Output Mesh

(c) Uniform boundary - Final Output Mesh

(d) L_2 Norm of Solution Error Convergence Plot

Figure 5.13: Summary of S-Shock Case Results

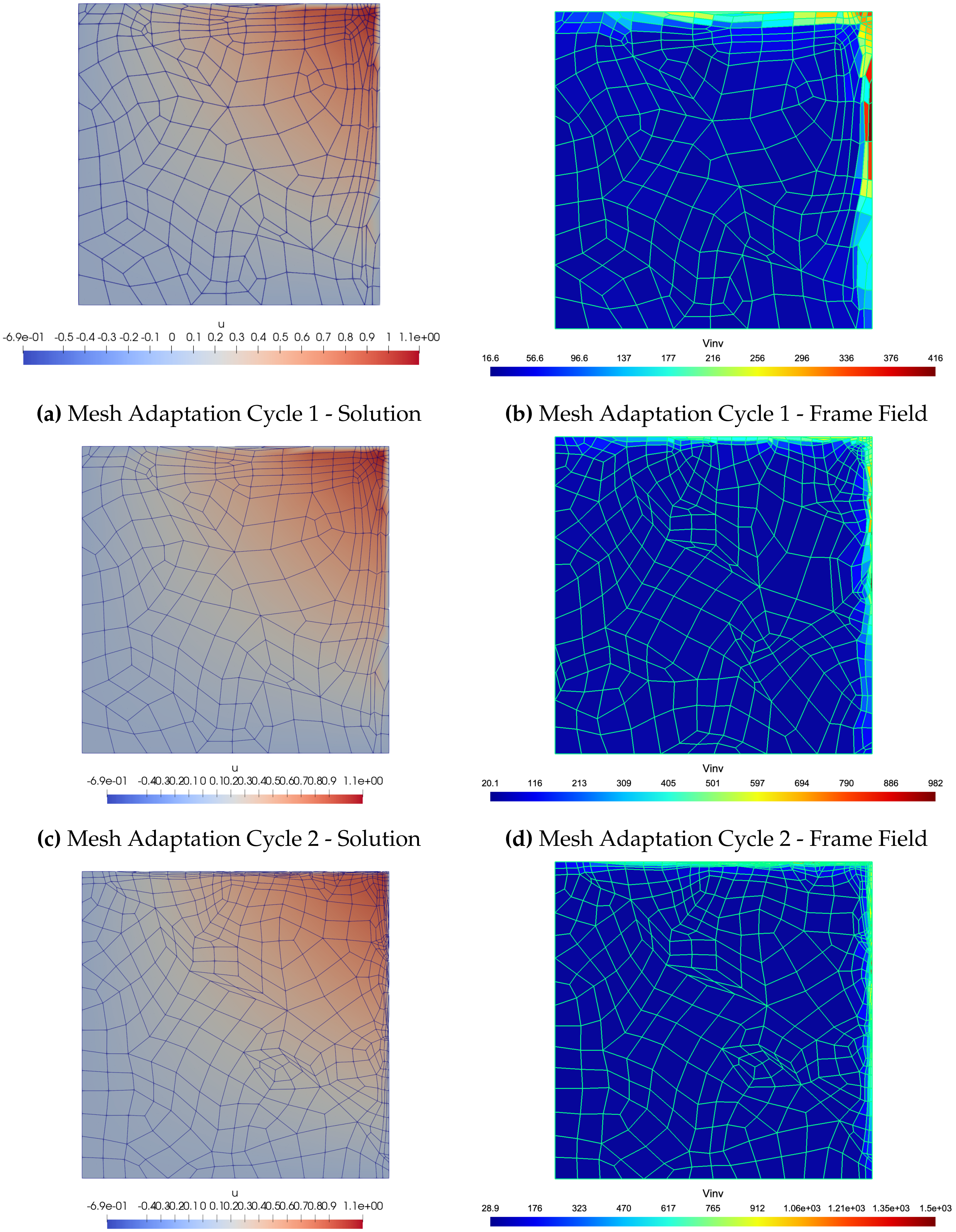
5.2.2 Boundary Layer Adaptation

The second discrete frame field case that is of interest is the boundary layer case. For this test case, a feature-based error estimator will be used. The manufactured solution is given by the following,

$$u(x,y) = \left(x + \frac{e^{\frac{x}{\epsilon}-1}}{1-e^{\frac{1}{\epsilon}}}\right)\left(y + \frac{e^{\frac{y}{\epsilon}-1}}{1-e^{\frac{1}{\epsilon}}}\right) \tag{5.6}$$

where ϵ is equal to 0.005 which has been used previously [77]. Similar to the s-shock test case, the boundary layer equation is solved within a domain of $\Omega = [0, 1]^2$ with a merge and splitting threshold of $L_{\mathrm{M}} \in [0.25, 2.25]$. Also, this test case is set up with 70 randomly distributed volume points and an arbitrary number of boundary points as defined by section 2.10. The uniqueness of this test case lies in its emphasis on attracting points near the top right corner of the domain and serves as a means to evaluate the mesh adaptation scheme's capability to generate anisotropic grids in regions of boundary layers. Figure 5.14 and 5.15 shows the results of the L_p-CVT algorithm after each mesh adaptation cycle for the moving boundary approach, and likewise, Figure 5.16 and 5.17 shows the results using the static boundary approach. One key observation of the convergence of the L_p-CVT energy gradient shown in Figure 5.15 e) and Figure 5.17 e) is that there are instabilities present in both the static and moving boundary because of the use of discrete frame fields. Despite their analogy to a continuous framework, achieving deep convergence poses a challenge, as noted in the preceding section. In terms of the total number of iterations, the moving boundary approach shows very quick convergence with about 350 iterations being the highest iteration count amongst the adaptation cycles. This is explained through the algorithm converging once detecting less than 2% of total edges requires splitting or merging as discussed in section 2.13. In contrast, the static boundary approach converges on the step length of the LBFGS algorithm, suggesting no further solution is possible in the search direction.

Figure 5.18 d) shows similar performance in the L_2 norm convergence plot where on some occasions, the static boundary would outperform the moving boundary method. However, the drawback is that the static boundary method does not allow for points moving on the boundary which would be critical for curved surfaces as will be demonstrated in the next section. Finally, both methods show a decrease in the L_2 norm convergence of about 2 orders of magnitude. The uniform mesh configuration shows the worst performance with about two times the total amount of DOFs in the last mesh adaptation cycle compared to standard static and moving boundary L_p-CVT.



(a) Mesh Adaptation Cycle 1 - Solution

(b) Mesh Adaptation Cycle 1 - Frame Field

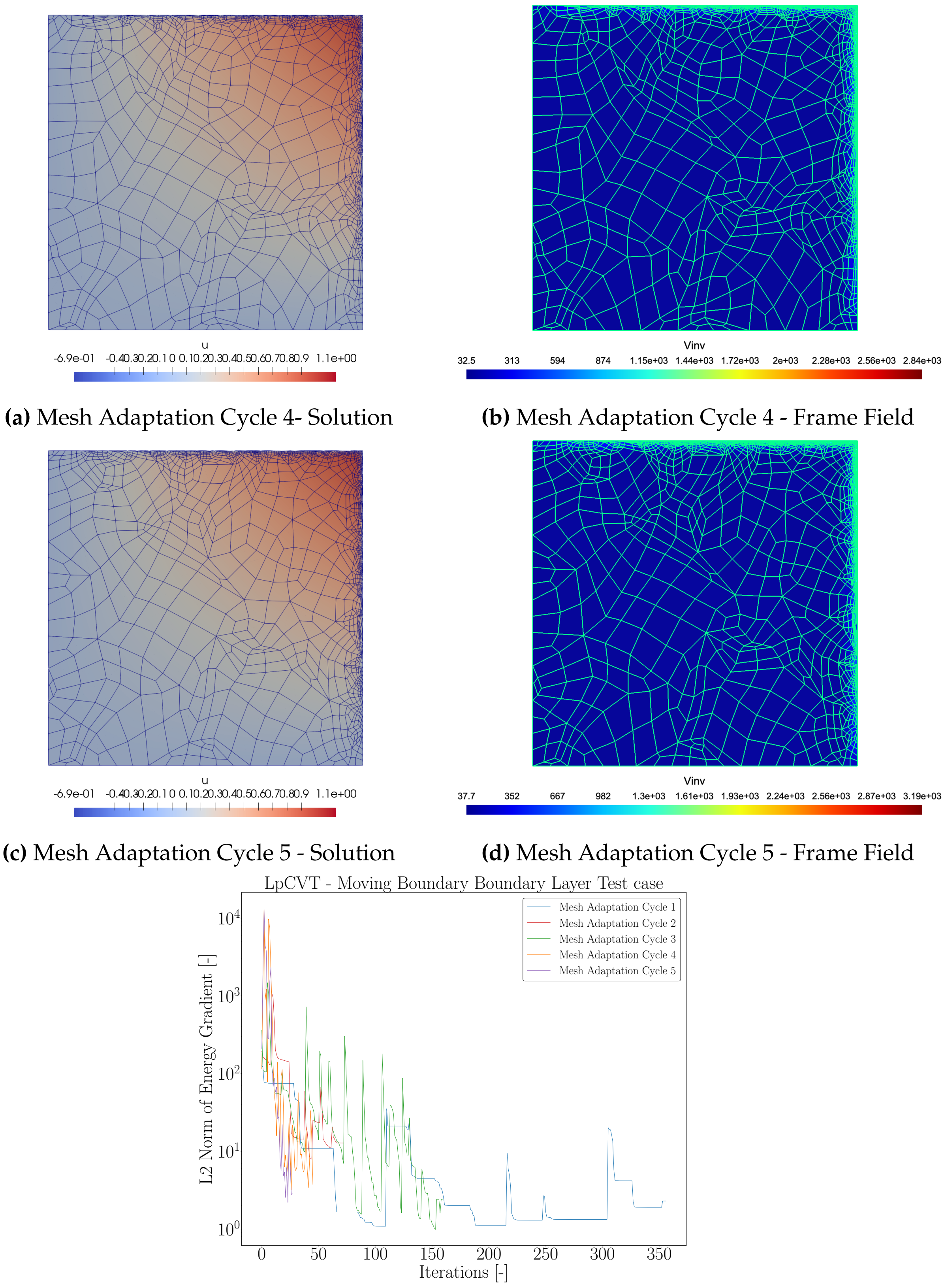
(c) Mesh Adaptation Cycle 2 - Solution

(d) Mesh Adaptation Cycle 2 - Frame Field

(e) Mesh Adaptation Cycle 3 - Solution

(f) Mesh Adaptation Cycle 3 - Frame Field

Figure 5.14: Summary of Boundary Layer Case Results Set 1 - Moving Boundary



(a) Mesh Adaptation Cycle 4- Solution

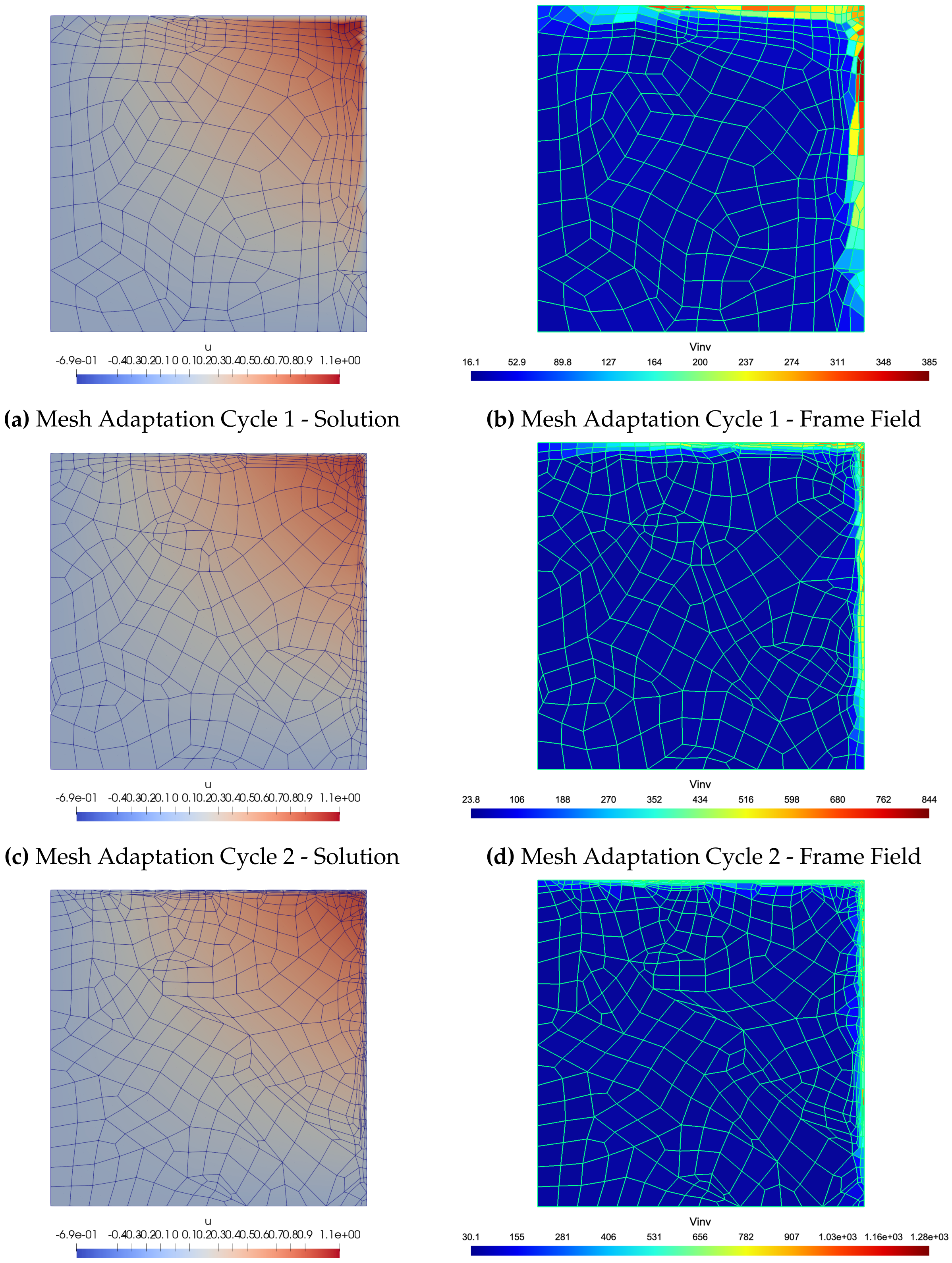
(b) Mesh Adaptation Cycle 4 - Frame Field

(c) Mesh Adaptation Cycle 5 - Solution

(d) Mesh Adaptation Cycle 5 - Frame Field

(e) L_p-CVT - L_2 norm Energy Gradient

Figure 5.15: Summary of Boundary Layer Case Results set 2 - Moving Boundary



(a) Mesh Adaptation Cycle 1 - Solution

(b) Mesh Adaptation Cycle 1 - Frame Field

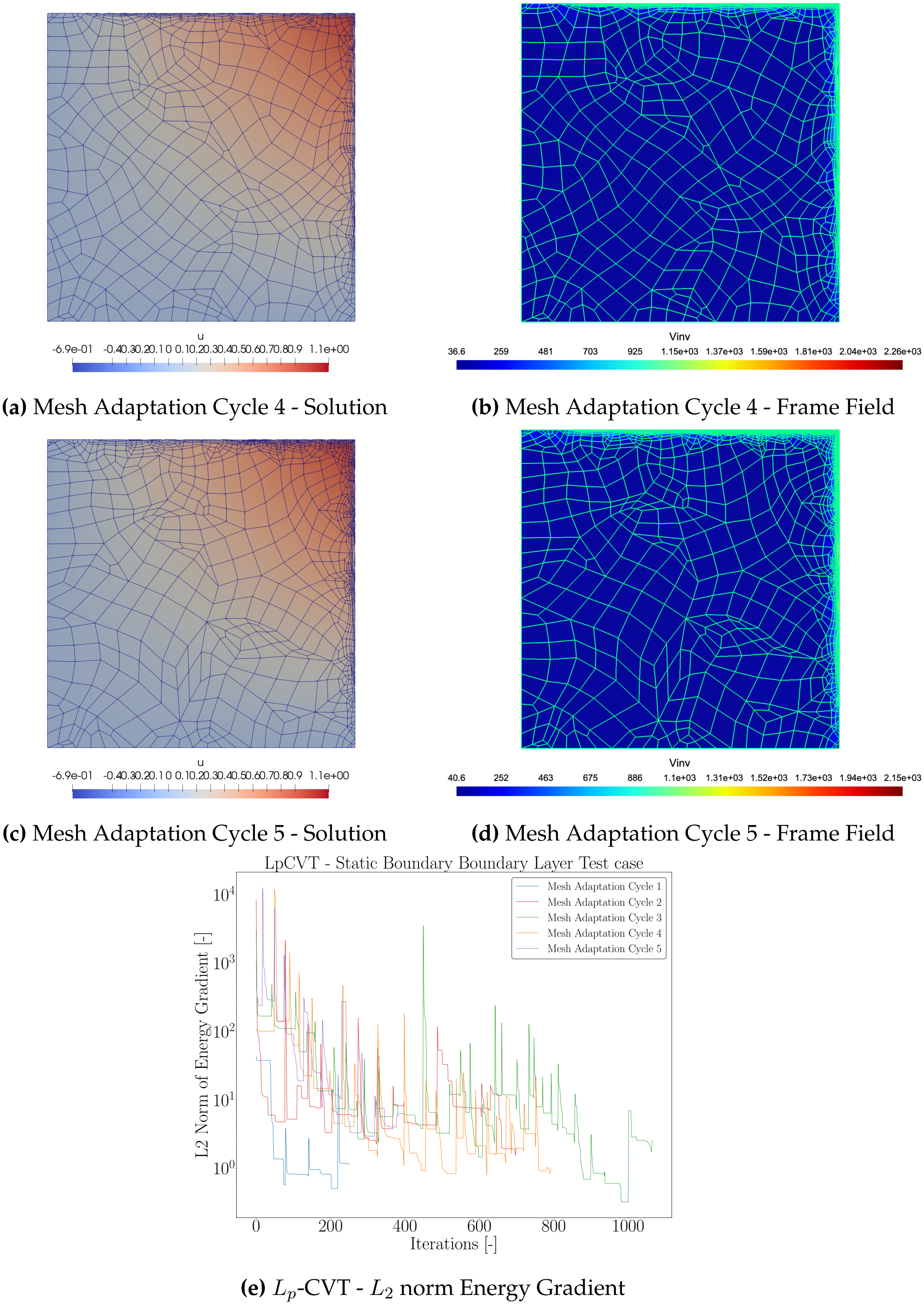
(c) Mesh Adaptation Cycle 2 - Solution

(d) Mesh Adaptation Cycle 2 - Frame Field

(e) Mesh Adaptation Cycle 3 - Solution

(f) Mesh Adaptation Cycle 3 - Frame Field

Figure 5.16: Summary of Boundary Layer Case Results Set 1 - Static Boundary



(a) Mesh Adaptation Cycle 4 - Solution

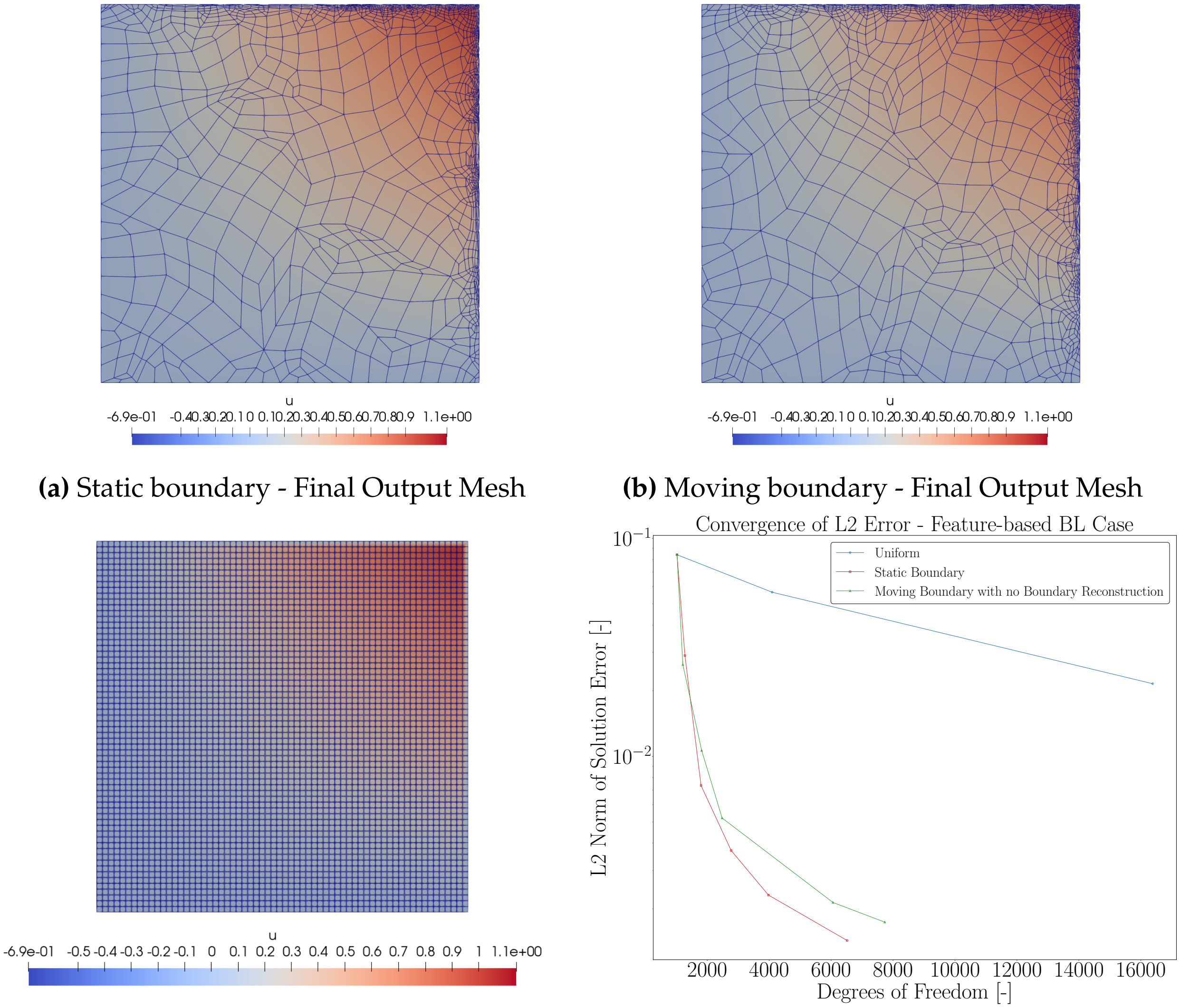
(b) Mesh Adaptation Cycle 4 - Frame Field

(c) Mesh Adaptation Cycle 5 - Solution

(d) Mesh Adaptation Cycle 5 - Frame Field

(e) L_p-CVT - L_2 norm Energy Gradient

Figure 5.17: Summary of Boundary Layer Case Results Set 2 - Static Boundary



(a) Static boundary - Final Output Mesh

(b) Moving boundary - Final Output Mesh

(c) Uniform boundary - Final Output Mesh

(d) L_2 Norm of Solution Error Convergence Plot

Figure 5.18: Summary of Boundary Layer Case Results

5.3 Euler Test Case

The flow solver test case is based on solving a two-dimensional Euler equation to test the capability of the boundary treatment applied to the L_p-CVT algorithm for real-world applications. However, due to the early maturity of the scheme, only a single adjoint-based example is explored to show the preliminary results of applying this scheme to a flow test case.

5.3.1 Transonic NACA0012 Steady State Test Case

This test case aims to resolve a shockwave on a NACA0012 airfoil set at an angle of attack of 1.25 with the inflow condition set to Mach 0.8 resulting in a transonic flow. The governing equation is the two-dimensional Euler's equation shown as follows,

$$\partial_x F + \partial_x G = 0 \tag{5.7}$$

where

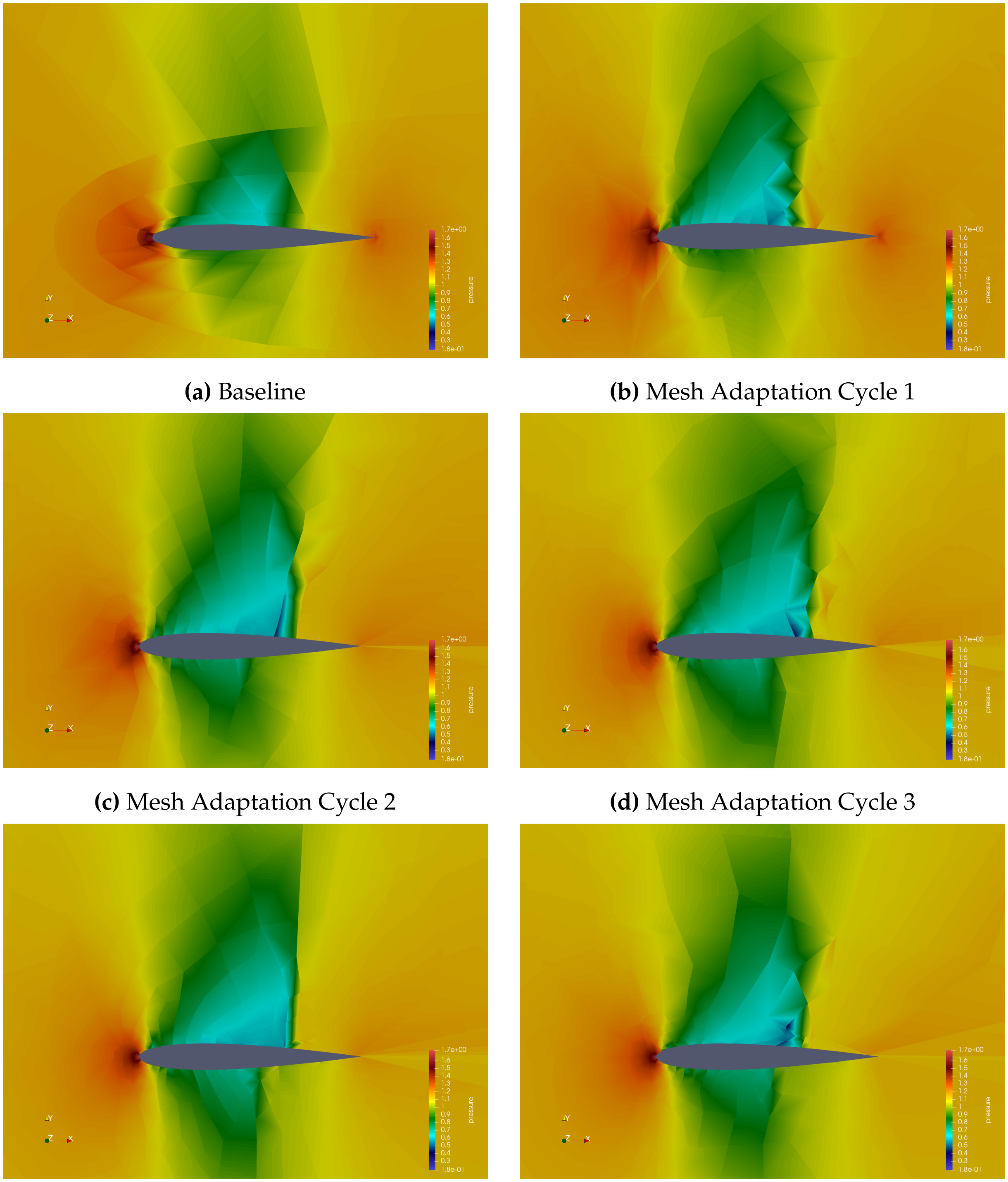
$$F = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ \rho u\,(e+p) \end{bmatrix}, G = \begin{bmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ v(\rho e + p) \end{bmatrix}, \tag{5.8}$$

where u and v are the velocity components in both horizontal and vertical directions, ρ is the density, p is the pressure and e is the internal energy component.

For this test case, an adjoint-based approach is used to evaluate the effectiveness of the L_p-CVT approach in adapting the initial mesh based on a drag-based functional. The solution is approximated using a first-order polynomial, or $p = 1$, and a steady-state solution is of interest. The pressure solutions based on the moving boundary approach are shown in Figure 5.19. Similarly, the results of the static boundary approach are shown in Figure 5.20. In addition to the L_p-CVT approach, a fixed-fraction method is also employed for comparison purposes of the drag values outputted using each scheme with a fixed-fraction of $f = 0.05$. The fixed-fraction method is a simple grid refinement strategy employed to refine local cells by quadrupling the number of cells within a single original cell. Hence, a single cell effectively becomes four cells in the following mesh adaptation cycle [39] if marked for refinement. The marking of cells to be refined is determined through the highest element-wise error in the domain and the portion of cells to be refined is based on the fixed-fraction parameter f as defined previously to be 5%. Note that the fixed-fraction method is only used as a reference in this test case in addition to the L_p-CVT method.

Five mesh adaptation cycle is targeted for this test case to investigate the convergence on the drag coefficient. The baseline grid is the same for all three methods, and it is a coarse grid composed of 560 cells. It can be seen that the baseline grid does a poor job of resolving the shockwave above the airfoil which is expected since a $p = 1$ solution with coarse grids above the airfoil does not capture well the flow features. At the first mesh adaptation cycle for both the static and moving boundary, the shockwave begins to appear suggesting that both methods are starting to converge towards the expected flow features. Interestingly, the static boundary approach yields a small shockwave under the airfoil while the moving boundary does not. This may be because the moving boundary approach allows the points to slide on the airfoil surface, allowing for more points to gather on the top side first to resolve the upper shockwave. In contrast, the static boundary approach has predefined and permanent airfoil point locations throughout all the mesh adaptation cycles, which adds a dependency on the original grid for solution convergence. This is especially apparent when looking at iterations 2 to 5 for the static boundary approach where the bottom shockwave is more and more apparent, and remains at the same location. On the contrary, the moving boundary approach slowly resolves a bottom shockwave as the adaptation cycles increase.

Figure 5.21 shows the final output mesh for each approach as well as the corresponding leading edge. In contrast to the fixed-fraction, for both the static and moving boundary approaches, the boundary points are projected on the exact airfoil surface during the quadrilateral recombination step resulting in a much better representation of the airfoil curvature. Figure 5.21 e) shows the leading edge of the fixed-fraction final output mesh which contains refined cells, however, most of them are linear, which does not help in resolving the curvature of the airfoil. As expected, Figure 5.21 a) shows a symmetric mesh configuration along the horizontal axis due to the static boundary configuration, however, this results in larger cells above the airfoil surface and very skewed and small cells near the leading edge which is a limitation of the static boundary approach. The distribution of the boundary points on the airfoil at the beginning of the mesh adaptation phase already predetermines the possible expected cell size that can be resolved near the airfoil. Hence, a uniform cell distribution is therefore observed above and below the airfoil, undermining the flexibility to resolve the target cell size and orientation if smaller and skewed cells are required in those regions as shown in Figure 5.21 b). Figure 5.21 c) shows that the leading edge of the moving boundary approach yields much smaller cells near the leading edge, and also, unsymmetric boundary distribution is observed which demonstrates added flexibility in resolving the target cell size as well as the original curvature of the airfoil throughout the mesh adaptation cycles. Lastly, the presented results are preliminary and hence the convergence of the functionals such as the lift and drag are not sufficiently mature to be presented. This will be part of future work.



(a) Baseline

(b) Mesh Adaptation Cycle 1

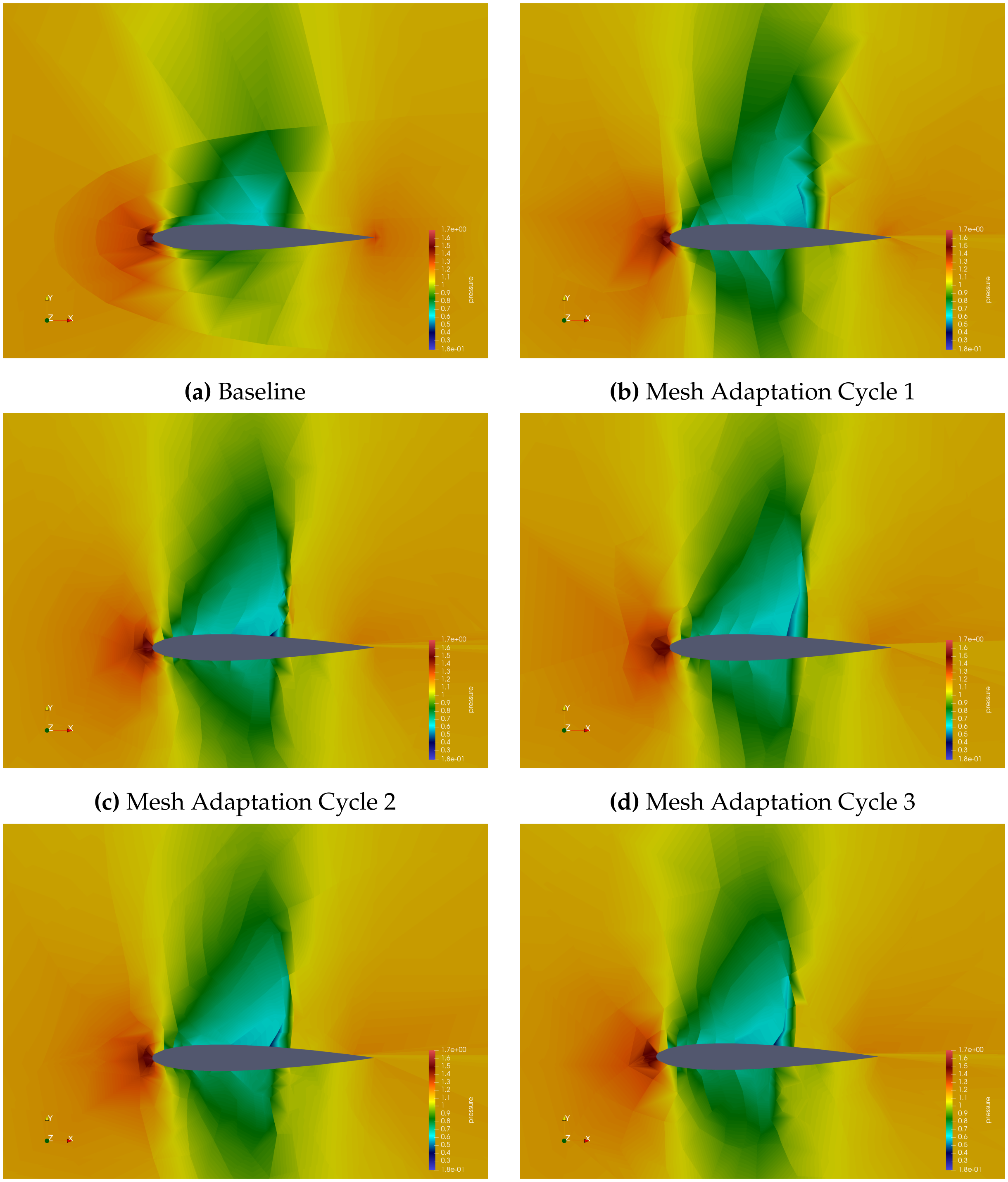
(c) Mesh Adaptation Cycle 2

(d) Mesh Adaptation Cycle 3

(e) Mesh Adaptation Cycle 4

(f) Mesh Adaptation Cycle 5

Figure 5.19: Moving Boundary L_p-CVT Mesh Adaptation Cycles



(a) Baseline

(b) Mesh Adaptation Cycle 1

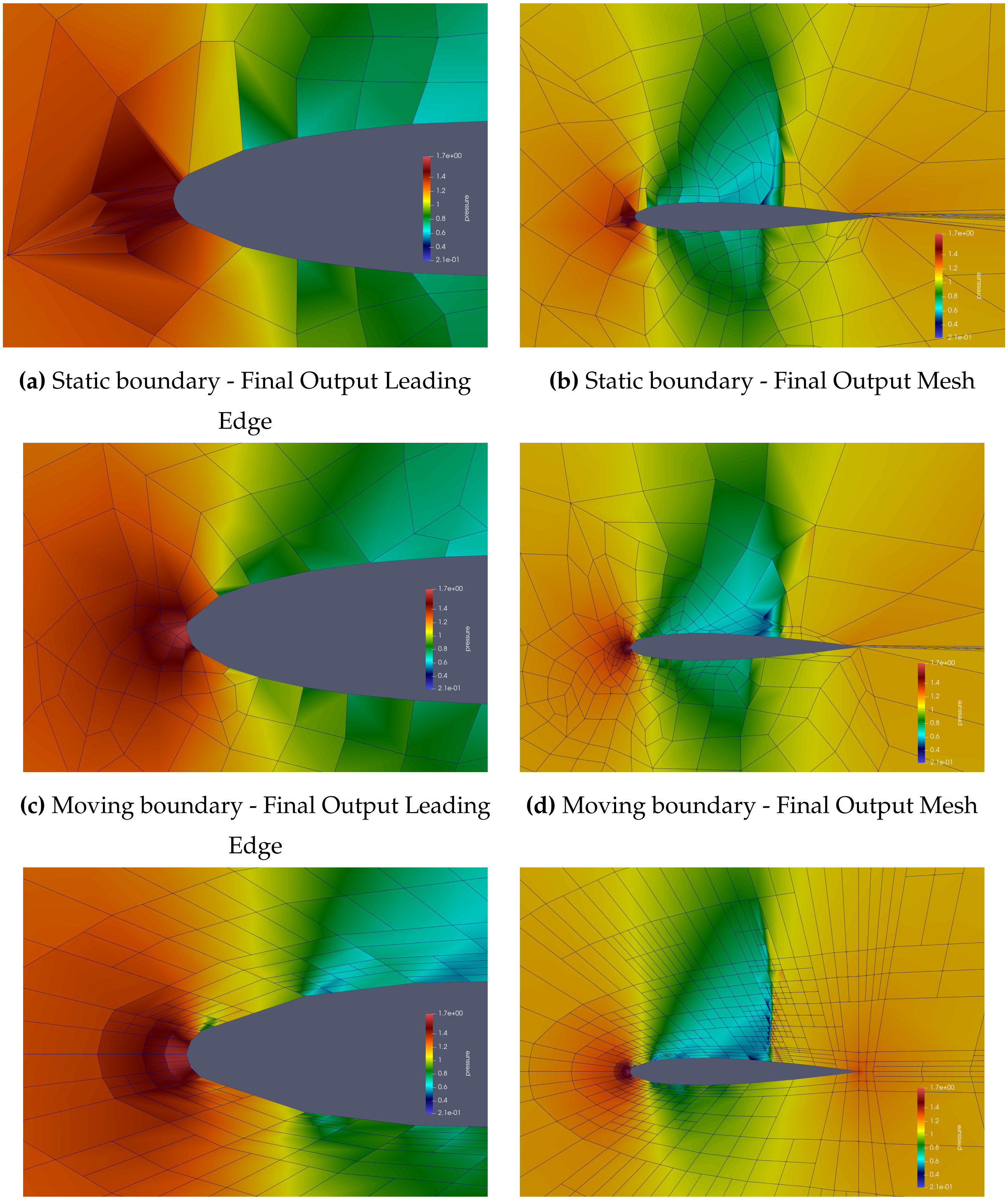
(c) Mesh Adaptation Cycle 2

(d) Mesh Adaptation Cycle 3

(e) Mesh Adaptation Cycle 4

(f) Mesh Adaptation Cycle 5

Figure 5.20: Static Boundary L_p-CVT Mesh Adaptation Cycles



(a) Static boundary - Final Output Leading Edge

(b) Static boundary - Final Output Mesh

(c) Moving boundary - Final Output Leading Edge

(d) Moving boundary - Final Output Mesh

(e) Fixed-Fraction Method - Final Output Leading Edge

(f) Fixed-Fraction Method - Final Output Mesh

Figure 5.21: Final Output Mesh Result Summary

Chapter 6

Conclusion

In conclusion, the moving boundary method demonstrates enhancements over the static boundary approach by providing flexibility at the boundary, allowing for the movement of points at specific locations of interest. This is demonstrated through the analytical metric field test cases where a similar final output mesh has been observed between the static and moving boundary approach, however, in the case of the moving boundary approach, a non-uniform starting boundary distribution can be used instead to yield similar results. In the case of more challenging test cases such as the quadratic analytical metric field, a simple boundary reconstruction technique has been introduced to recover the incorrect energy gradient computed at the boundary due to clipped Voronoi cells. This technique has proven to be useful in analytical test cases, however, poor results were observed when used with discrete frame fields. The s-shock test case showed that the simplified boundary reconstruction technique can be wrongly employed when drastic changes in the discrete frame fields around the boundary are present. Nonetheless, it was shown that the regular moving boundary approach showed satisfactory results when compared to the static boundary approach, achieving a three- to four-order reduction in the convergence. It was also shown through the boundary layer test case that instabilities in the energy gradient convergence are present for both the moving and static boundary approach, suggesting that discrete frame fields are more likely to converge based on the merging and splitting threshold criteria or step length convergence rather than on gradient convergence. Finally, the transonic flow solver test case showed that the moving boundary method can add flexibility to resolve curvatures on surfaces while performing mesh adaptation based on different target cell sizes, however, this could potentially alter the original surface, yielding a different geometry at the final output mesh unless a higher-order representation of the grid is employed.

6.1 Future Work

Future work can be focused on improving the boundary reconstruction scheme to reconstruct the boundary based on a local cell area rather than averaging the metric length required for yielding a quasi-unit mesh over the entire boundary. Additional checks can also be done on the final output mesh to verify if the original geometry of the surfaces present in the domain is still valid after performing boundary point movements. Additionally, more flow test cases should be performed to evaluate the maturity of the moving boundary approach, and convergence of the functional such as lift and drag should be investigated once convergence is achieved. Lastly, a high-order version of the L_p-CVT can be explored which would likely start with the static boundary approach, and then, progress toward the moving boundary approach. The high-order version of this algorithm should benefit the adjoint-based mesh adaptation as the coefficient of lift or drag can be much better resolved following a curvature in the elements near the airfoil surface which would help track the flow features that are naturally driven by the shape of the airfoil. Moreover, the high-order representation of the grid would better maintain the airfoil geometry.

Bibliography

[1] M. L. Hosain and R. B. Fdhila, “Literature review of accelerated CFD simulation methods towards online application,” *Energy Procedia*, vol. 75, pp. 3307–3314, Aug. 2015. [Online]. Available: https://doi.org/10.1016/j.egypro.2015.07.714

[2] A. Cary, J. Chawner, E. Duque, W. Gropp, B. Kleb, R. Kolonay, E. Nielsen, and B. Smith, “Realizing the vision of CFD in 2030,” *Computing in Science &amp Engineering*, vol. 24, no. 1, pp. 64–70, Jan. 2022. [Online]. Available: https://doi.org/10.1109/mcse.2021.3133677

[3] Z. Wang, K. Fidkowski, R. Abgrall, F. Bassi, D. Caraeni, A. Cary, H. Deconinck, R. Hartmann, K. Hillewaert, H. Huynh, N. Kroll, G. May, P.-O. Persson, B. van Leer, and M. Visbal, “High-order CFD methods: current status and perspective,” *International Journal for Numerical Methods in Fluids*, vol. 72, no. 8, pp. 811–845, Jan. 2013. [Online]. Available: https://doi.org/10.1002/fld.3767

[4] 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,” *International Journal for Numerical Methods in Fluids*, vol. 32, no. 6, pp. 725–744, Mar. 2000. [Online]. Available: https://doi.org/10.1002/(sici)1097-0363(20000330)32:6⟨725::aid-fld935⟩3.0.co;2-4

[5] D. Ait-Ali-Yahia, G. Baruzzi, W. G. Habashi, M. Fortin, J. Dompierre, and M.-G. Vallet, “Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent CFD. part II. structured grids,” *International Journal for*

Numerical Methods in Fluids, vol. 39, no. 8, pp. 657–673, 2002. [Online]. Available: https://doi.org/10.1002/fld.356

[6] J. Dompierre, M.-G. Vallet, Y. Bourgault, M. Fortin, and W. G. Habashi, "Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent CFD. part III. unstructured meshes," *International Journal for Numerical Methods in Fluids*, vol. 39, no. 8, pp. 675–702, 2002. [Online]. Available: https://doi.org/10.1002/fld.357

[7] F. Alauzet and A. Loseille, "A decade of progress on anisotropic mesh adaptation for computational fluid dynamics," *Computer-Aided Design*, vol. 72, p. 13–39, Mar. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.cad.2015.09.005

[8] W. Huang, L. Kamenski, and J. Lang, "A new anisotropic mesh adaptation method based upon hierarchical a posteriori error estimates," *Journal of Computational Physics*, vol. 229, no. 6, p. 2179–2198, Mar. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2009.11.029

[9] R. Verfürth, "A posteriori error estimation and adaptive mesh-refinement techniques," *Journal of Computational and Applied Mathematics*, vol. 50, no. 1–3, p. 67–83, May 1994. [Online]. Available: http://dx.doi.org/10.1016/0377-0427(94)90290-9

[10] A. Belme, F. Alauzet, and A. Dervieux, "An a priori anisotropic goal-oriented error estimate for viscous compressible flow and application to mesh adaptation," *Journal of Computational Physics*, vol. 376, p. 1051–1088, Jan. 2019. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2018.08.048

[11] A. Belme, "Unsteady aerodynamics and adjoint method," Ph.D. dissertation, Université Nice Sophia Antipolis, 2011.

[12] A. Loseille, A. Dervieux, and F. Alauzet, "Fully anisotropic goal-oriented mesh adaptation for 3d steady euler equations," *Journal of Computational*

Physics, vol. 229, no. 8, p. 2866–2897, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2009.12.021

[13] O. C. Zienkiewicz and J. Z. Zhu, "The superconvergent patch recovery and a posteriori error estimates. part 1: The recovery technique," *International Journal for Numerical Methods in Engineering*, vol. 33, no. 7, p. 1331–1364, May 1992. [Online]. Available: http://dx.doi.org/10.1002/nme.1620330702

[14] O. C. Zienkiewicz and J. Z. Zhu, "The superconvergent patch recovery and a posteriori error estimates. part 2: Error estimates and adaptivity," *International Journal for Numerical Methods in Engineering*, vol. 33, no. 7, p. 1365–1382, May 1992. [Online]. Available: http://dx.doi.org/10.1002/nme.1620330703

[15] V. Dolejší, "Anisotropic mesh adaptation for finite volume and finite element methods on triangular meshes," *Computing and Visualization in Science*, vol. 1, no. 3, p. 165–178, Nov. 1998. [Online]. Available: http://dx.doi.org/10.1007/s007910050015

[16] V. Dolejší, "Anisotropic hp-adaptive method based on interpolation error estimates in the lq-norm," *Applied Numerical Mathematics*, vol. 82, p. 80–114, Aug. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.apnum.2014.03.003

[17] V. Dolejší, G. May, and A. Rangarajan, "A continuous hp-mesh model for adaptive discontinuous galerkin schemes," *Applied Numerical Mathematics*, vol. 124, p. 1–21, Feb. 2018. [Online]. Available: http://dx.doi.org/10.1016/j.apnum.2017.09.015

[18] A. Loseille and F. Alauzet, "Continuous mesh framework part i: Well-posed continuous interpolation error," *SIAM Journal on Numerical Analysis*, vol. 49, no. 1, p. 38–60, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1137/090754078

[19] A. Loseille and F. Alauzet, "Continuous mesh framework part ii: Validations and applications," *SIAM Journal on Numerical Analysis*, vol. 49, no. 1, p. 61–86, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1137/10078654X

[20] N. Ringue and S. Nadarajah, “Optimization-based anisotropic hp-adaptation for high-order methods,” in *23rd AIAA Computational Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, Jun. 2017. [Online]. Available: http://dx.doi.org/10.2514/6.2017-3101

[21] M. Ainsworth and J. Oden, “A posteriori error estimation in finite element analysis,” *Computer Methods in Applied Mechanics and Engineering*, vol. 142, no. 1–2, p. 1–88, Mar. 1997. [Online]. Available: http://dx.doi.org/10.1016/S0045-7825(96)01107-3

[22] C. Roy, “Strategies for driving mesh adaptation in cfd (invited),” in *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*. American Institute of Aeronautics and Astronautics, Jan. 2009. [Online]. Available: http://dx.doi.org/10.2514/6.2009-1302

[23] A. Balan, M. A. Park, S. L. Wood, W. K. Anderson, A. Rangarajan, D. P. Sanjaya, and G. May, “A review and comparison of error estimators for anisotropic mesh adaptation for flow simulations,” *Computers amp; Fluids*, vol. 234, p. 105259, Feb. 2022. [Online]. Available: http://dx.doi.org/10.1016/j.compfluid.2021.105259

[24] E. F. D’Azevedo and R. B. Simpson, “On optimal interpolation triangle incidences,” *SIAM Journal on Scientific and Statistical Computing*, vol. 10, no. 6, p. 1063–1075, Nov. 1989. [Online]. Available: http://dx.doi.org/10.1137/0910064

[25] E. F. D’Azevedo and R. B. Simpson, “On optimal triangular meshes for minimizing the gradient error,” *Numerische Mathematik*, vol. 59, no. 1, p. 321–348, Dec. 1991. [Online]. Available: http://dx.doi.org/10.1007/BF01385784

[26] G. Karniadakis and S. Sherwin, *Spectral/hp Element Methods for Computational Fluid Dynamics*. Oxford University Press, Jun. 2005. [Online]. Available: http://dx.doi.org/10.1093/acprof:oso/9780198528692.001.0001

[27] D. Panozzo, E. Puppo, M. Tarini, and O. Sorkine-Hornung, "Frame fields: Anisotropic and non-orthogonal cross fields additional material," *Proceedings of the ACM TRANSACTIONS ON GRAPHICS (PROCEEDINGS OF ACM SIGGRAPH*, 2014.

[28] O. Diamanti, A. Vaxman, D. Panozzo, and O. Sorkine-Hornung, "Designing n-polyvector fields with complex polynomials," in *Computer Graphics Forum*, vol. 33, no. 5. Wiley Online Library, 2014, pp. 1–11.

[29] M. J. Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial differential equations," *Journal of computational Physics*, vol. 53, no. 3, pp. 484–512, 1984.

[30] M. J. Berger and A. Jameson, "Automatic adaptive grid refinement for the euler equations," *AIAA journal*, vol. 23, no. 4, pp. 561–568, 1985.

[31] G. P. Warren, W. K. Anderson, J. L. Thomas, and S. T. Krist, "Grid convergence for adaptive methods," 1991. [Online]. Available: https://api.semanticscholar.org/CorpusID:120939509

[32] J. Peraire, M. Vahdati, K. Morgan, and O. C. Zienkiewicz, "Adaptive remeshing for compressible flow computations," *Journal of computational physics*, vol. 72, no. 2, pp. 449–466, 1987.

[33] J. Peraire, J. Peiro, and K. Morgan, "Adaptive remeshing for three-dimensional compressible flow computations," *Journal of Computational Physics*, vol. 103, no. 2, pp. 269–285, 1992.

[34] P. George, F. Hecht, and M. Vallet, "Creation of internal points in voronoi's type method. control adaptation," *Advances in Engineering Software and Workstations*, vol. 13, no. 5–6, p. 303–312, Sep. 1991. [Online]. Available: http://dx.doi.org/10.1016/0961-3552(91)90034-2

[35] R. Becker and R. Rannacher, “A feed-back approach to error control in finite element methods: Basic analysis and examples,” 1996. [Online]. Available: https://api.semanticscholar.org/CorpusID:10696092

[36] D. A. Venditti and D. L. Darmofal, “Adjoint error estimation and grid adaptation for functional outputs: Application to quasi-one-dimensional flow,” *Journal of Computational Physics*, vol. 164, no. 1, p. 204–227, Oct. 2000. [Online]. Available: http://dx.doi.org/10.1006/jcph.2000.6600

[37] D. A. Venditti and D. L. Darmofal , “Grid adaptation for functional outputs: Application to two-dimensional inviscid flows,” *Journal of Computational Physics*, vol. 176, no. 1, p. 40–69, Feb. 2002. [Online]. Available: http://dx.doi.org/10.1006/jcph.2001.6967

[38] D. A. Venditti and D. L. Darmofal, “Anisotropic grid adaptation for functional outputs: application to two-dimensional viscous flows,” *Journal of Computational Physics*, vol. 187, no. 1, p. 22–46, May 2003. [Online]. Available: http://dx.doi.org/10.1016/S0021-9991(03)00074-3

[39] K. MacLean and S. Nadarajah, “Anisotropic mesh generation and adaptation for quads using the l-cvt method,” *Journal of Computational Physics*, vol. 470, p. 111578, Dec. 2022. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2022.111578

[40] P.-O. Persson and G. Strang, “A simple mesh generator in matlab,” *SIAM Review*, vol. 46, no. 2, p. 329–345, Jan. 2004. [Online]. Available: http://dx.doi.org/10.1137/S0036144503429121

[41] L. P. Chew, “Constrained delaunay triangulations,” in *Proceedings of the third annual symposium on Computational geometry - SCG ’87*, ser. SCG ’87. ACM Press, 1987. [Online]. Available: http://dx.doi.org/10.1145/41958.41981

[42] C. Geuzaine and J. Remacle, “Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities,” *International Journal for Numerical*

Methods in Engineering, vol. 79, no. 11, p. 1309–1331, May 2009. [Online]. Available: http://dx.doi.org/10.1002/nme.2579

[43] J. Steger and R. Sorenson, "Automatic mesh-point clustering near a boundary in grid generation with elliptic partial differential equations," *Journal of Computational Physics*, vol. 33, no. 3, p. 405–410, Dec. 1979. [Online]. Available: http://dx.doi.org/10.1016/0021-9991(79)90165-7

[44] B. Delaunay, S. Vide, A. Lamémoire, and V. De Georges, "Bulletin de l'academie des sciences de l'urss," *Classe des sciences mathématiques et naturelles*, vol. 6, pp. 793–800, 1934.

[45] O. R. Musin, "Properties of the delaunay triangulation," in *Proceedings of the thirteenth annual symposium on Computational geometry*, 1997, pp. 424–426.

[46] L. Chen and J.-c. Xu, "Optimal delaunay triangulations," *Journal of Computational Mathematics*, pp. 299–308, 2004.

[47] N. P. Weatherill, "Delaunay triangulation in computational fluid dynamics," *Computers & Mathematics with Applications*, vol. 24, no. 5-6, pp. 129–150, 1992.

[48] B. Lévy and Y. Liu, "Lp Centroidal Voronoi Tessellation and its applications," *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 1–11, Jul. 2010. [Online]. Available: https://doi.org/10.1145/1778765.1778856

[49] Y. Liu, W. Wang, B. Lévy, F. Sun, D.-M. Yan, L. Lu, and C. Yang, "On centroidal voronoi tessellation—energy smoothness and fast computation," *ACM Transactions on Graphics (ToG)*, vol. 28, no. 4, pp. 1–17, 2009.

[50] M. Iri, K. Murota, and T. Ohya, "A fast voronoi-diagram algorithm with applications to geographical optimization problems," in *System Modelling and Optimization: Proceedings of the 11th IFIP Conference Copenhagen, Denmark, July 25–29, 1983*. Springer, 1984, pp. 273–288.

[51] W. J. Gordon and C. A. Hall, “Construction of curvilinear co-ordinate systems and applications to mesh generation,” *International Journal for Numerical Methods in Engineering*, vol. 7, no. 4, p. 461–477, Jan. 1973. [Online]. Available: http://dx.doi.org/10.1002/nme.1620070405

[52] W. J. Gordon and L. C. Thiel, “Transfinite mappings and their application to grid generation,” *Applied Mathematics and Computation*, vol. 10–11, p. 171–233, Jan. 1982. [Online]. Available: http://dx.doi.org/10.1016/0096-3003(82)90191-6

[53] J. F. Thompson, F. C. Thames, and C. Mastin, “Automatic numerical generation of body-fitted curvilinear coordinate system for field containing any number of arbitrary two-dimensional bodies,” *Journal of Computational Physics*, vol. 15, no. 3, p. 299–319, Jul. 1974. [Online]. Available: http://dx.doi.org/10.1016/0021-9991(74)90114-4

[54] J. K. Hodge, A. L. Stone, and T. E. Miller, “Numerical solution for airfoils near stall in optimized boundary-fitted curvilinear coordinates,” *AIAA Journal*, vol. 17, no. 5, p. 458–464, May 1979. [Online]. Available: http://dx.doi.org/10.2514/3.61155

[55] J. WHITE, “Elliptic grid generation with orthogonality and spacing control on an arbitrary number of boundaries,” in *21st Fluid Dynamics, Plasma Dynamics and Lasers Conference*. American Institute of Aeronautics and Astronautics, Jun. 1990. [Online]. Available: http://dx.doi.org/10.2514/6.1990-1568

[56] G. Moretti, “Grid generation using classical techniques,” *NASA. Langley Research Center Numerical Grid Generation Tech.*, 1980.

[57] P. R. Eiseman, “Grid generation for fluid mechanics computations,” *Annual Review of Fluid Mechanics*, vol. 17, no. 1, p. 487–522, Jan. 1985. [Online]. Available: http://dx.doi.org/10.1146/annurev.fl.17.010185.002415

[58] A.-S. W. Lindberg, T. M. Jørgensen, and V. A. Dahl, “Linear, transfinite and weighted method for interpolation from grid lines applied to oct images,”

Applied Soft Computing, vol. 68, p. 293–302, Jul. 2018. [Online]. Available: http://dx.doi.org/10.1016/j.asoc.2018.03.031

[59] Y. C. Liou, *Journal of Scientific Computing*, vol. 13, no. 1, p. 105–114, 1998. [Online]. Available: http://dx.doi.org/10.1023/A:1023260812163

[60] K. A. Hoffmann and S. T. Chiang, "Computational fluid dynamics volume i," *Engineering education system*, 2000.

[61] R. KUMAR, "Elliptic grid generation for naca0012 airfoil."

[62] N. N. Sørensen, "Hypgrid2d. a 2-d mesh generator," 1998.

[63] A. P. Kuprat, C. W. Mastin, and A. Khamayseh, "Boundary orthogonality in elliptic grid generation," 1998. [Online]. Available: https://api.semanticscholar.org/CorpusID:123368319

[64] P. Frey, "Yams a fully automatic adaptive isotropic surface remeshing procedure," Ph.D. dissertation, Inria, 2001.

[65] W. Cao, "Anisotropic measures of third order derivatives and the quadratic interpolation error on triangular elements," *SIAM Journal on Scientific Computing*, vol. 29, no. 2, p. 756–781, Jan. 2007. [Online]. Available: http://dx.doi.org/10.1137/050634700

[66] T. P. Minka, "Old and new matrix algebra useful for statistics," *See www. stat. cmu. edu/minka/papers/matrix. html*, vol. 4, 2000.

[67] T. C. Baudouin, J.-F. Remacle, E. Marchandise, J. Lambrechts, and F. Henrotte, "Lloyd's energy minimization in the Lp norm for quadrilateral surface mesh generation," *Engineering with Computers*, vol. 30, no. 1, pp. 97–110, Oct. 2012. [Online]. Available: https://doi.org/10.1007/s00366-012-0290-x

[68] D. Ekelschot, M. Ceze, A. Garai, and S. M. Murman, “Robust metric aligned quad-dominant meshing using lp centroidal voronoi tessellation,” in *2018 AIAA Aerospace Sciences Meeting*. American Institute of Aeronautics and Astronautics, Jan. 2018. [Online]. Available: http://dx.doi.org/10.2514/6.2018-1501

[69] D. Shi-Dong and S. Nadarajah, “Full-space approach to aerodynamic shape optimization,” *Computers amp; Fluids*, vol. 218, p. 104843, Mar. 2021. [Online]. Available: http://dx.doi.org/10.1016/j.compfluid.2021.104843

[70] The CGAL Project, *CGAL User and Reference Manual*, 5.6 ed. CGAL Editorial Board, 2023. [Online]. Available: https://doc.cgal.org/5.6/Manual/packages.html

[71] J. Remacle, J. Lambrechts, B. Seny, E. Marchandise, A. Johnen, and C. Geuzainet, “Blossom-quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm,” *International Journal for Numerical Methods in Engineering*, vol. 89, no. 9, p. 1102–1119, Feb. 2012. [Online]. Available: http://dx.doi.org/10.1002/nme.3279

[72] D. Shi-Dong, S. Nadarajah, J. Brillon, D. Blais, Keigan MacLean, Pranshul Thakur, C. Pethrick, A. Cicchino, and M. Tatarelli, “dougshidong/philip: Philip version 2.0.0,” 2022. [Online]. Available: https://zenodo.org/record/6600853

[73] T. Trilinos Project Team, *The Trilinos Project Website*.

[74] Z. Wang, K. Fidkowski, R. Abgrall, F. Bassi, D. Caraeni, A. Cary, H. Deconinck, R. Hartmann, K. Hillewaert, H. Huynh, N. Kroll, G. May, P. Persson, B. van Leer, and M. Visbal, “High-order cfd methods: current status and perspective,” *International Journal for Numerical Methods in Fluids*, vol. 72, no. 8, p. 811–845, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1002/fld.3767

[75] W. H. Reed and T. R. Hill, “Triangular mesh methods for the neutron transport equation,” Los Alamos Scientific Lab., N. Mex.(USA), Tech. Rep., 1973.

[76] A. Khursheed, *High-Order Elements*. Boston, MA: Springer US, 1999, pp. 99–110. [Online]. Available: https://doi.org/10.1007/978-1-4615-5201-7_5

[77] A. Rangarajan, A. Balan, and G. May, “Mesh optimization for discontinuous galerkin methods using a continuous mesh model,” *AIAA Journal*, vol. 56, no. 10, pp. 4060–4073, 2018.

[78] A. Balan, M. Woopen, and G. May, “Adjoint-based hp -adaptivity on anisotropic meshes for high-order compressible flow simulations,” *Computers amp; Fluids*, vol. 139, p. 47–67, Nov. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.compfluid.2016.03.029