Title
Discontinuous Galerkin Methods on Moving Domains with Large Deformations

Author
Wang, Luming

Publication Date
2015

Peer reviewed|Thesis/dissertation

Discontinuous Galerkin Methods on Moving Domains with Large Deformations

by

Luming Wang

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Applied Mathematics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Per-Olof S. Persson, Chair
Professor John A. Strain
Professor Sanjay Govindjee

Spring 2015

Discontinuous Galerkin Methods on Moving Domains with Large Deformations

Abstract

Discontinuous Galerkin Methods on Moving Domains with Large Deformations

by

Luming Wang

Doctor of Philosophy in Applied Mathematics

University of California, Berkeley

Professor Per-Olof S. Persson, Chair

We present two different numerical approaches for solving compressible flows on moving domains with high-order accuracy. The approaches are base on discontinuous Galerkin (DG) methods and are particularly designed for addressing the large deformation problems as the domain moves.

A moving-mesh technique is first introduced to improve the mesh quality with the domain deforming. The technique moves the mesh nodes by DistMesh algorithm[60] and locally changes the mesh topology by flipping edges or faces, which can be applied in both 2D and 3D. Moreover, some local density control operations are also developed to add or remove the mesh nodes to change the mesh adaptivity.

Our first numerical scheme is formulated on a space-time framework using a nodal DG discretization on space-time domains with appropriate numerical fluxes for the first and the second-order terms, respectively. The scheme is implicit, and we solve the resulting non-linear systems using a parallel Newton-Krylov solver. Along with the numerical scheme, two efficient algorithms for constructing globally conforming space-time slab meshes are given, based on our moving-mesh technique.

The second approach employs DG discreatization with arbitrary-Eulerian-Lagrangian (ALE) framework by solving equations based on smooth mappings. An efficient local L^2 projection is used for transferring solutions when mesh topology change happens.

We test our two approaches by a number of numerical cases in both 2D and 3D. The tests involve convergence tests as well as simulations of laminar flows, which shows that the proposed methods achieve high-order accuracy and are able to handle problems with complex geometric motions.

To My Family

I love you all.

Contents

Contents **ii**

List of Figures **iv**

List of Tables **viii**

1 Introduction **1**
1.1 Prior Work 2
1.2 Overview 4

2 Governing Equations **6**
2.1 Compressible Navier-Stokes Equations 6
2.2 Isentropic Flow 7
2.3 Boundary Conditions 8

3 Moving-Mesh Strategy **9**
3.1 Mesh Node Smoothing 9
3.2 Local Mesh Topology Change 11
3.3 Mesh Adaptivity 13
3.4 The algorithm 15

4 Space-Time Discontinuous Galerkin Methods **17**
4.1 Space-Time Formulation 17
4.2 Numerical Scheme 18
4.2.1 Discretization of the Euler Equations 18
4.2.2 Discretization of the Viscosity Terms 21
4.2.3 Newton-Krylov Solver 23
4.3 Space-Time Mesh Generation for 2D Problems 24
4.3.1 Basic Idea 24
4.3.2 Local Triangulation of Prisms 26
4.3.3 Diagonal Matching and the Global Algorithm 29
4.4 Space-Time Mesh Generation for 3D Problems 34
4.4.1 An Alternative Algorithm for 2D Problems 34

4.4.2 Generalization of Prisms, Lateral Faces and Diagonals 37
4.4.3 Extension to 3D Problems 39

5 Arbitrary Lagrangian-Eulerian Discontinuous Galerkin Methods with Local L^2 Projections **43**
5.1 Arbitrary Lagrangian-Eulerian Formulation 43
5.2 Numerical Scheme 45
5.2.1 Discretization 45
5.2.2 Geometric Conservation Law 46
5.2.3 Temporal Integration 46
5.3 Local L^2 Projections 47
5.3.1 Formulation 48
5.3.2 Implementation 50

6 Numerical Results **54**
6.1 Euler Vortex 54
6.1.1 2D Case 54
6.1.2 3D Case 58
6.2 Spinning Cross 60
6.3 Pitching Tandem Airfoils 63
6.4 Airfoil with a Deploying Spoiler 65
6.5 Double Vertical Axis Wind Turbines 66
6.6 Rotating 3D Ellipsoid 74

7 Conclusion & Future Work **76**

Bibliography **77**

List of Figures

3.1 Force-based smoothing. The plot shows the net force exerted on one node. (Image from [61]) . . . 10
3.2 Local topology change in 2D.(Image from [61]) . . . 12
3.3 Change of local element connectivity in 3D. The numbers of old and new elements are shown above the arrows. . . . 12
3.4 Density Control Operations in 2D. These operations are used to control the local mesh density by adding or removing nodes. . . . 14
3.5 Density Control Operations in 3D. These operations are used to control the local mesh density by adding or removing nodes. . . . 14

4.1 An example for node distribution of DG method. Here, $p = 2$ and 6 nodes are chosen within each element K. (Image from [61]) . . . 19
4.2 Space-Time Mesh Generation. The left figure illustrates two mesh layers at time t and $t + \Delta t$, and the right figure shows a corresponding 3D space-time mesh between the two layers. The blue faces show a cross-section of the tetrahedral mesh. . . . 25
4.3 Tetrahedral Triangulation in 2D. The left plot illustrates a valid triangulation for an element without edge flips, and the right plot shows a triangulation for a pair of elements with a flipped edge. . . . 25
4.4 All the valid triangulations of a triangular prism (left) and of a quadrangular prism (right). The sets below each triangulation show the values of the corresponding sign functions. . . . 28
4.5 Examples of triangulated prisms corresponding to local density control operations. 29
4.6 Two examples of updating paths. Each triangle represents a triangular prism and each quadrilateral represents a quadrangular prism. The yellow element is the root prism V, the green elements are prisms at state either 1 or 2, and the red elements are already triangulated, i.e., at state 3. The purple elements denote an updating path directed by the black arrows. The corresponding case number that each purple element belongs to is also shown. The example path on the left ends when it returns to V, and the example path on the right ends with a triangular prism belonging to Case 1. . . . 32
4.7 Triangulation of polygonal prisms with an additional interior node. . . . 35

4.8 An example of an invalid triagulation for a non-convex polygon. The point marked in red is the inserted node. 36
4.9 'Prisms' and 'Lateral Faces' in 2D, 3D and 4D. In each case, the right is a 'prism' and the left is one of its 'lateral faces'. 39

5.1 The mapping between the reference domain and the physical domain in the ALE framework. 44
5.2 Moving Meshes for a Spinning Cross without Topology Change. Three sample plots are given to show the mesh motion under the ALE framework. 48
5.3 An Example of Global Remeshing. The 3rd plot overlaps mesh 1 with mesh 2. . 50
5.4 An illustration of the process in algorithm 5.1. Phase I: Select blue nodes to move; Phase II: Update interior nodes by force-based smoothing; Phase III: Select pairs of elements in red for flipping; Phase IV: Change the local connectivity within each pair of red elements. 51
5.5 Local element splitting in 2D. 52
5.6 Tetrahedral splitting for 3D operations. In each plot, the triangulation of the middle polygon is on the left and the corresponding 3D triangulation is on the right. The red cross is the intersection between the middle polygon and the connecting line between the top and the bottom nodes. 52

6.1 An illustration of the moving meshes. The five blues nodes are rotated about the center of the domain in a rigid manner, which induces other vertex movements and local element flipping. Note that θ is the degrees that the blue nodes have rotated about the center. 55
6.2 Space-Time Convergence Test for the Euler Vortex Problem. The top three plots are samples of space-time meshes with $\Delta t = h = 0.3125$ and $p = 3$. Below each mesh, the corresponding solutions are shown in pressure fields. The pressure plots uses the 'hot' colormap in Matlab with range $[68.5, 71.6]$. The bottom plot shows the convergence results for $p = 1$, 2 and 3. 57
6.3 Euler-Vortex Convergence Plot for the ALE Method with Local L^2 Projections. 58
6.4 Convergence Test for the 3D Euler Vortex Problem. The upper left and middle plots are two sample meshes at the initial and the final time, respectively. These meshes are generated on a $10 \times 10 \times 10$ cube. The blue faces show a cross-section of the tetrahedral mesh and the green faces are the outside surfaces of the cube. The red nodes are rotated rigidly. The right plot shows some sample pressure isosurfaces, which uses the 'jet' colormap in Matlab with range $[2.2377, 2.5871]$. . 59
6.5 Initial Spatial Mesh of a Spinning Cross. The left zoom-in figure is for the upper right part of the initial mesh on the right. 60
6.6 Examples of Spatial Meshes and Space-Time Mesh Slabs. Note that to better illustrate our space-time mesh structure, we rescale the thickness of our space-time mesh slabs, which is not equal to the real Δt we used for the simulation. . 61

6.7 Space-Time DG solutions for Compressible Navier-Stokes flow in a domain with a spinning cross (entropy). It uses the reverse 'hot' colormap in Matlab with range $[17.85, 18.05]$. 62
6.8 ALE DG solutions for Compressible Navier-Stokes flow in a domain with a spinning cross (entropy). It uses the reverse 'hot' colormap in Matlab with range $[18.02, 18.32]$. 63
6.9 Schematics of the pitching tandem airfoil problem. 64
6.10 Sample Spatial Meshes of Two Pitching Tandem Airfoils. 64
6.11 Numerical results for the pitching tandem airfoils. There are solution fields for both space-time and ALE methods (entropy of the flow at 4 time instances). It uses the 'jet' colormap in Matlab with range $[17.79, 18.18]$. 66
6.12 The drag and lift forces on the pitching tandem NACA0012 airfoils as a function of time for both the space-time and the ALE methods. 67
6.13 Spatial Meshes of the Airfoil with a Deploying Spoiler. 68
6.14 Space-Time DG Solutions for Compressible Navier-Stokes flow around an airfoil with a deploying spoiler. It uses the reverse 'hot' colormap in Matlab with range $[17.79, 18.18]$. 69
6.15 Drag and lift coefficients around the NACA0012 airfoil with a deploying spoiler as a function of time. 69
6.16 Schematics of the double vertical axis wind turbines. (Image from [38]) 70
6.17 Unstructured Mesh for Double VAWTs. The initial mesh is showed on the top, where all the edge flipping operations happen in the area colored in red. In order to show the mesh motion, three zoom-in plots are placed on the bottom for the area circled by a yellow window in the top plot. 71
6.18 Numerical results for double VAWTs. Sample solutions are plotted for the y-component of the velocity and for the vorticity field, respectively. The plots for y-component of the velocity used reverse RdBu colormap in Python's matplotlib module. The color range is $[0.03656, 0.14624]$. The plots for vorticity used RdBu colormap with the color range is $[-50, 50]$. 72
6.19 Sample 3D Meshes for Spinning Ellipsoid Problem. The surface mesh of the inside ellipsoid and outside sphere is colored in green. The blue faces show a cross-section of the spherical mesh. 73
6.20 Sample Solutions for Spinning Ellipsoid Problem. The surface of the spinning ellipsoid is painted in green and we plot the Mach solution on the entropy isosurface. It uses the 'jet' colormap in Matlab with range $[0, 0.25]$. 75

List of Algorithms

3.1 Moving-Mesh Startegy 15
4.1 Space-Time Mesh Generation with Path Marching 33
4.2 Space-Time Mesh Generation with Insertion of Additional Nodes 38
5.1 Discontinuous Galerkin ALE Method with Local L^2 Projections 51

List of Tables

4.1 The summary of sign function value combination for local triangulations. 28
4.2 Summary of the geometric properties for prisms in 2D, 3D and 4D. 39

5.1 Butcher's Array for the DIRK3 scheme. 47

Acknowledgments

First of all, I would like to thank my advisor, Professor Per-Olof Persson for motivating and encouraging me to work on this interesting research topic. I would not complete this Ph.D. dissertation without his insightful guidance and kind support. To me, he is not only an outstanding advisor in academia, but a good friend that I can frankly share my thoughts and feelings. It has been a fairly pleasant experience to work with him and he helps me find my interest and strength, improve my personality and build my future career.

I also feel grateful to my other committee members, Professor John Strain and Professor Sanjay Govindjee to provide me with valuable feedback and suggestions. In particular, I appreciate every piece of great advice from Professor Strain on teaching sections, passing the qualifying exam and revising journal papers. I would thank Professor Jon Wilkening for joining the committee of my qualifying exam, Professor Ming Gu for guiding me on the work of numerical linear algebra, and Professor Alexandre Chorin and Professor James Sethian for leading our mathematics group in Lawrence Berkeley Laboratory and making us like a big family.

I would acknowledge my colleagues Meire Fortunato, Bradley Froehle, Samuel Kanner and Christopher Melgaard. I enjoy our collaboration and discussion, where many of my research ideas were inspired. I would also thank my colleagues Jue Chen, Jeff Donatelli, Long Jin, Anna Lieb, Weihua Liu, Fei Lu, Matthias Morzfeld, Benjamin Preskill, Wei Qi, Qingchun Ren, Robert Saye and Ethan Van Andel. I'm glad to meet you all during my graduate years.

I must express my greatest gratitude to my wife Yanan Pei and my other family members. Whenever I feel disappointed or exhausted, I know they are always my most solid and reliable support, and encourage me keeping moving forward.

Finally, I would like to thank the support from the AFOSR Computational Mathematics program under grant FA9550-10-1-0229; the Alfred P. Sloan foundation; the Lawrence Berkeley National Laboratory and the National Energy Research Scientic Computing Center funded by the Director, Oce of Science, Computational and Technology Research, U.S. Department of Energy under Contract No. DE-AC02-05CH11231; and the Simons Fellowship.

Chapter 1

Introduction

Many practical applications involve time-varying geometries such as rotor-stator flows, flapping flight or fluid-structure interactions. Comparing with building real models and carrying out physical experiments, high fidelity numerical simulations are acknowledged to be a promising alternative approach for studying those engineering problems due to its low time and manufacturing cost. For this reason, in the field of computational fluid mechanics, turbulent flow simulation on deforming domains becomes one of the most popular research topics.

However, such problems face a few challenges with respect to the numerical methods. First, since the physical domains are moving, the computational meshes must be generated in a way that is able to track the geometric motions. There are mainly two classes of methods to address this issue: embedded domain method and body-fitted method. The former one depends on a Cartesian grid and then embeds the irregular domain into the grid. This approach allows the computational mesh not necessarily to be aligned with the domain boundary and benefits from the fact that the generation of a Cartesian grid is trivial. Many popular schemes are invented following these ideas such as the embedded boundary/cut-cell methods [14, 30], the immersed boundary method [39, 63], and the fictitious domain methods [28, 82]. These schemes are often simple and highly efficient, but somewhat difficult to modify for high-order accuracy close to the boundaries and for thin boundary layers. On the other hand, the body-fitted method employs unstructured meshes and automatically remove the difficulty of cell cutting around the boundary. The central concern about these strategies is how to move the mesh in order to guarantee that the computational mesh can always fit with good mesh quality even for complicated geometry motions.

Besides the meshing issues, since these simulations always involve large-scale computation, an efficient and robust numerical solver is important in order to reduce the computational cost and provide the sufficiently accurate test results. Therefore, it is widely believed that high-order numerical methods will become standard in the future numerical engineering software. A number of high-order numerical methods have been proposed for solving flow problems including spectral methods, streamline-upwind/Petrov-Galerkin (SUPG) and high-order volume methods. Among these, discontinuous Galerkin (DG) methods have received

much attention during the last decade due to their ability to produce stable and high-order accurate discretizations of conservation laws on fully unstructured meshes [12, 55]. In particular for challenging fluid problems, the low dissipation of the DG schemes make them ideal for the simulation of turbulent flows with complex vortical structures and non-linear interactions.

Various solutions have been developed for compressible/incompressible flows on deformable domains, but most of them are only reliable for solving problems with mild deformation. It still lacks an effective numerical approach to deal with the cases when the domain deformation is large and/or complex. In the following work, we focus on the study of high-order DG methods on fully unstructured meshes, in particular with the goal of addressing moving-domain problems with large deformation.

1.1 Prior Work

The so-called space-time framework are fully consistent discretizations that allow for arbitrary changes of the domain in both space and time [47]. These methods essentially treats the time-dependency with the same technique as the spatial terms, but exploiting the causal nature of the equations to improve the efficiency. Some of the early work on space-time methods include [33, 35], where a Galerkin/least square finite element method was introduced for solving classical elastodynamics problems. Extensions to flow problems on moving domains, with SUPG stabilization, where developed in [3, 49]. A priori and a posteriori error estimates for a space-time finite element discretization for linear second order hyperbolic equations were presented in [37].

General formulations and analyses of space-time DG methods were presented in [79, 80]. The methods have been used for a number of practical applications, for example in [40] where a space-time DG method was demonstrated on several aerodynamic applications. Formulations for the Oseen equations where developed in [78], and for two-fluid flows in [71]. Since the space-time DG formulations lead to fully implicit discretizations, many previous authors have studied specialized solution techniques. For example, [41] provided a pseudo-time stepping method to deal with the time evolution, and an h-multigrid solver was introduced in [42]. Also, space-time hybridizable discontinuous Galerkin (HDG) methods were recently proposed in [66, 67].

Most work on space-time DG methods have focused on spatial meshes that do not undergo connectivity changes throughout the time evolution. The corresponding space-time meshes are then essentially composed of extruded prism-elements, possibly deforming in time. Some previous work have explored time adaptivity, for example in [48] where a space-time finite volume method with nonuniform time stepping was developed. However, to handle moving domain problems with large deformations, fully unstructured space-time meshes are required since the spatial mesh topology has to be modified during the simulation. Previous work on fully unstructured space-time meshes includes [65], which proposed a space-time finite volume method on unstructured meshes generated by a standard 3D mesh generator. A

so-called tent-pitcher space-time mesh generator based on an advancing front method was introduced in [76, 77] with various extensions in [1, 23, 74]. Finally, in [8] space-time meshes were generated by connecting spatial meshes using a Delaunay approach, which required some additional techniques to eliminate sliver elements.

As an alternative, the Arbitrary Lagrangian-Eulerian (ALE) method [21, 32] is able to produce high-order accurate solutions to a wide range of problems associated with body-fitted grid motion. It originates from solid mechanics, where the Lagrangian framework is widely used to track the material deformation. However, for fluid problems, this approach may result in severe mesh distortions, which will eventually break the simulation down. Instead, ALE method combines the advantages of both Eulerian and Lagrangian frameworks and has been proved to be powerful for simulating turbulent flows on moving domains. The formulations of ALE method are based on a time-varying mapping and a deforming grid that conforms to the domain boundary, which is compensated for by a modification of the equations. Since the first time when ALE formulation was proposed, it has been developed to solve various problems numerically: a finite element method with an ALE framework was presented in [34] for incompressible viscous flows; an application to incompressible hyperelasticity can be found in [87]; the ALE analysis for free surface flow was studied in [9] and an analysis of the nonlinear stability was discussed in [24].

The discontinuous Galerkin method along with an ALE framework for compressible viscous flows was introduced in [46]. [57] implemented a mapping-based ALE formulation with a compact discontinuous Galerkin (CDG) methods and a novel way to enforce the so-called geometric conservation law (GCL) using an additional set of ODEs. [52] applied this approach to two-phase flow problems. [51] computes conservation laws on moving meshes using a discontinuous Galerkin Spectral Element Method (DGSEM). [50] derived an ALE-DG scheme which ensures the satisfaction of the GCL condition. ALE-DG methods for fluid-structure interaction were proposed in [27, 83].

However, a main drawback with the ALE approach is that it is difficult to deform the mesh without inverting the elements as the domain undergoes large deformations. That is, in order for the deformation mapping to be smooth, the mesh topology must be fixed which means that the initial element connectivities have to remain unchanged throughout the time evolution. This restriction can be severe for large or complex deformations. Remeshing is then commonly employed to maintain well-shaped elements, but in order to transfer the solutions between the original and the recomputed mesh, careful treatment is needed to obtain accuracy and stability. Many interpolation techniques have been proposed, as well as several topology-change strategies. [81] provides an ALE formulation with a new fixed grid approach. [69] delivered an adaptive remeshing strategy for flow simulations. Several effective mesh motion techniques for large deformation problems in 3D were shown in [5]. A technique for incorporating mesh adaption was presented in [15]. [53, 2] studied an ALE framework with edge flipping techniques and [36, 29] developed an edge-based finite volume solver with mesh topology changes. However, in general the accuracy is easily reduced by additional numerical errors introduced from solution transferring, and thus so far it appears that satisfactory high-order accurate ALE methods for large boundary deformations have

not yet been developed.

1.2 Overview

In this thesis, we improve the two numerical approaches mentioned in section 1.1, respectively, for solving the compressible flow problems on moving domains with large deformation. Our ideas both originate from a novel moving mesh strategy, which is able to handle arbitrary geometric motions efficiently and robustly. This moving mesh strategy involves a number of operations including mesh node smoothing, edge swapping and mesh adaptivity. The main advantage is that the proposed algorithm only changes the mesh topology locally, which provides opportunities to largely simplify the numerical schemes for a space-time or an ALE framework.

First, based on the moving mesh strategy, we present a space-time discontinuous Galerkin discretization of the compressible Navier-Stokes equations and a novel fully unstructured space-time mesh generation procedure. Our space-time discretizations are derived from the compact discontinuous Galerkin method [54], they are higher-dimensional and fully implicit, and we solve the resulting nonlinear systems of equations using efficient parallel Newton-Krylov solvers [58, 56]. For space-time mesh generation, unlike the previous work outlined above, our procedures improve the spatial meshes by the moving-mesh algorithm rather than regenerate meshes from scratch at each time step. Moreover, based on this particular spatial-mesh improvement strategy, our space-time meshes are fully unstructured and generated efficiently and robustly. Two algorithms for space-time mesh generation are developed, both only depending on the combinatoric properties of how the vertices are connected. The first one is a depth-first search algorithm, mainly for 2D problems. Since no additional nodes are inserted besides those on the spatial meshes, its computational cost for solving the discretized systems is kept at a minimum. The second algorithm allows the addition of a few new nodes, which works well for both 2D and 3D problems. Lastly, the order of accuracy of the resulting space-time discontinuous Galerkin scheme can be arbitrarily high in both space and time, provided suitable curved meshes can be generated.

Next, we propose a simple combined approach based on a nodal DG formulation and an ALE framework. The discretization is based on the work in [57], which solves a modified version of compressible Navier-Stokes equations using a DG scheme. To deal with large deformation problems, we rely on a local L^2 projection operator to transfer the solution between the old and the new meshes. The standard L^2 projections are straight-forward to formulate and have many desirable properties, but the implementation is complicated and costly for high-dimensional unstructured meshes, which limits their practical applications. But we again take advantage of the locality properties of our mesh adjustment during the domain motion and only carry out the corresponding L^2 projections locally, which significantly simplifies its implementation. The numerical results show that our local L^2 projection is highly efficient and can maintain high-order accuracy, even with frequent mesh topology changes, compared to other interpolation techniques.

The rest of this thesis is organized as follows: chapter 2 introduces the governing equations which we aim to solve; chapter 3 describes the details of our moving mesh algorithm in both 2D and 3D; chapter 4 includes the general formulation of our space-time DG scheme and two algorithms for space-time mesh generation; chapter 5 discusses the ALE framework with the local L^2 projection; chapter 6 presents numerical results for our two numerical methods, where the high-order accuracy and the ability to address large deformation are tested with various numerical simulations; finally, brief conclusions and suggestions for future work are given in chapter 7.

Chapter 2

Governing Equations

2.1 Compressible Navier-Stokes Equations

We consider the compressible Navier-Stokes equations [55] as follows:

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i}(\rho u_i) &= 0 \\ \frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_j}(\rho u_i u_j + P\delta_{ij}) &= \frac{\partial}{\partial x_j}(\tau_{ij}) \quad \text{for } \; i = 1, 2, 3 \\ \frac{\partial}{\partial t}(\rho E) + \frac{\partial}{\partial x_j}(u_j(\rho E + P)) &= \frac{\partial}{\partial x_j}(u_i \tau_{ij} - \Theta_j) \end{aligned} \tag{2.1}$$

with some appropriate boundary conditions imposed on the domain boundary and initial conditions at $t = 0$. The viscous stress tensor τ_{ij} and heat flux Θ_i are given by

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\delta_{ij} \frac{\partial u_k}{\partial x_k} \right) \tag{2.2}$$

and

$$\Theta_i = -\frac{\mu}{Pr} \frac{\partial}{\partial x_i} \left(E + \frac{P}{\rho} - \frac{1}{2} u_k^2 \right), \tag{2.3}$$

where δ_{ij} is the Kronecker delta function, μ is the viscosity coefficient and Pr is the Prandtl number, which we assume to be constant 0.72.

In these equations, ρ is the fluid mass density, ρE is the total energy, u_i is the component of the velocity along each spatial direction, respectively. The quantity P is the pressure.

Under the assumption of the ideal gas, pressure P can be written as [45]:

$$P = \rho R T \tag{2.4}$$

where R is the ideal gas constant which can be expressed as the difference between the heat capacities at constant pressure C_P and at constant volume C_V. Denote e as the internal energy of the fluid and γ as the adiabatic gas constant. Then based on the relations

$$T = \frac{e}{C_V} = \frac{1}{C_V}(E - \frac{1}{2}u_k^2) \quad \text{and} \quad \gamma = \frac{C_P}{C_V} \tag{2.5}$$

we can derive from Eq 2.4 that

$$P = (\gamma - 1)\rho\left(E - \frac{1}{2}u_k^2\right), \tag{2.6}$$

Here, we assume γ to be the theoretical value of a diatomic ideal gas 1.4.

For simplicity, we will write Eq 2.1 in the following conservation form:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F}^{\text{inv}}(\boldsymbol{u}) = \nabla \cdot \boldsymbol{F}^{\text{vis}}(\boldsymbol{u}, \nabla \boldsymbol{u}), \tag{2.7}$$

where the solution vector $\boldsymbol{u}$, inviscid flux $\boldsymbol{F}^{\text{inv}}$ and viscous flux $\boldsymbol{F}^{\text{vis}}$ are given by

$$\boldsymbol{u} = \begin{pmatrix} \rho \\ \rho u_i \\ \rho E \end{pmatrix}, \qquad \boldsymbol{F}_j^{\text{inv}}(u) = \begin{pmatrix} \rho u_j \\ \rho u_i u_j + P\delta_{ij} \\ u_j(\rho E + p) \end{pmatrix}, \qquad \boldsymbol{F}_j^{\text{vis}}(u, \nabla \boldsymbol{u}) = \begin{pmatrix} 0 \\ \tau_{ij} \\ \tau_{ij} u_i - \Theta_j \end{pmatrix}. \tag{2.8}$$

If we zero out the viscous term (i.e. the right hand side of Eq 2.7), we obtain the Euler equations, which we will use for some of the convergence tests.

In the numerical tests, some physical quantities are used for visualization. We define the vorticity $\boldsymbol{\omega}$, entropy s and Mach number M as

$$\boldsymbol{\omega} = \nabla \times \boldsymbol{u}, \quad s = \frac{P}{\rho^\gamma} \quad \text{and} \quad M = \sqrt{\frac{\rho}{\gamma p}}\|\boldsymbol{u}\|_2. \tag{2.9}$$

2.2 Isentropic Flow

In some of our applications, we solve a simplified version of the compressible Navier-Stokes equations instead of Eq 2.1 in order to reduce the computational cost. The version assumes that the entropy s is constant throughout the simuation. That is, from Eq 2.9, we have the relation

$$P = s\rho^\gamma. \tag{2.10}$$

Since the pressure P can be evaluated without knowing the energy E according to Eq 2.10, we can remove the energy equations from Eq 2.1 and only solve for flow density and velocities. In this way, we reduce the size of the nonlinear system and thus have fewer degrees of freedom to consider.

Physically, this modification provides a good approximation for incompressible flow and it can be seen as an artificial compressibility method. [19] suggests that under some assumptions, as the Mach number approaches zero, the solution of the isentropic flow will converge to that of the incompressible Navier-Stokes equations:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{1}{\rho} \nabla P + \mu \Delta \boldsymbol{u} \tag{2.11}$$

$$\nabla \cdot \boldsymbol{u} = 0 \tag{2.12}$$

2.3 Boundary Conditions

In order to make the system of differential equations well-posed, we impose proper boundary conditions on the domain boundary. From [11], we consider two different types in our numerical tests.

First, when we simulate compressible flows around moving bodies, we use an adiabatic viscous wall condition on the walls. This condition requires that at the boundary, the velocity of the flow is exactly the same as that of the moving wall, i.e.

$$\boldsymbol{u} = \boldsymbol{u}_{\text{wall}}. \tag{2.13}$$

There is also no transfer of heat or mass between the wall and the fluid.

Second, in theory, many flow problems are defined on an infinite domain. However, computationally, the meshes can only be generated for a finite area. For this reason, we impose an artificial far-field 'boundary condition' on the outside mesh boundary so that we can compute the flow as if there is no outside boundary. We choose the so-called free-stream quantities by specifying density ρ_∞, velocity $\boldsymbol{u}_\infty$ and pressure P_∞, and then set the solution $\vec{\boldsymbol{u}}_\infty$ on the boundary as

$$\boldsymbol{u} = \begin{pmatrix} \rho_\infty \\ \rho_\infty \boldsymbol{u}_\infty \\ \frac{P_\infty}{\gamma - 1} + \frac{1}{2} \rho_\infty \|\boldsymbol{u}_\infty\|^2 \end{pmatrix}. \tag{2.14}$$

Note that we also use this free-stream solution ρ_∞, $\boldsymbol{u}_\infty$ and P_∞ as the initial condition in most of our flow simulations.

Chapter 3

Moving-Mesh Strategy

For body-fitted methods, as the physical domain deforms, the mesh has to be moved accordingly. In order to maintain high element qualities throughout the geometric motion, several local mesh operations are introduced to adjust the mesh. More specifically, denote $\Omega^t \subset \mathbb{R}^d$ as the time-varying domain at time t and $\mathcal{T}_t^h$ as the triangulation of Ω^t at time t with characteristic mesh size h. At the very beginning, an initial triangulation $\mathcal{T}_0^h$ of Ω^0 is generated using any standard unstructured mesh generation technique. For the next time step Δt, the triangulation $\mathcal{T}_{\Delta t}^h$ of $\Omega^{\Delta t}$ is obtained by performing operations on the previous triangulation $\mathcal{T}_0^h$. By repeating this process iteratively, we can generate a sequence of spatial meshes $\{\mathcal{T}_0^h, \mathcal{T}_{\Delta t}^h, \mathcal{T}_{2\Delta t}^h, \ldots, \mathcal{T}_T^h\}$ for all time steps. This chapter will talk about the details of these mesh operations.

3.1 Mesh Node Smoothing

At each time step, as the first step of our moving-mesh technique, we move all the boundary nodes rigidly according to the prescribed geometry motion. This approach is sufficient for our examples, but in general it might be necessary to, for example, redistribute the boundary nodes if the boundary deformations are large. The element qualities generally decrease after the boundary nodes are moved, and thus it is important to smooth the interior mesh nodes and then improve the mesh quality.

There are many approaches for mesh smoothing including spring-based methods [88], radial basis functions [17] and PDE-based methods [72]. For our approach, we choose the DistMesh scheme [60], which is believed to be one of the most efficient and robust approaches for mesh node smoothing.

The DistMesh scheme belong to the category of spring-based methods. As illustrated in figure 3.1, the movement of interior nodes is driven by repulsive forces along each attached edge, which depend on the edge length l and a user-specified equilibrium length l_0. It is

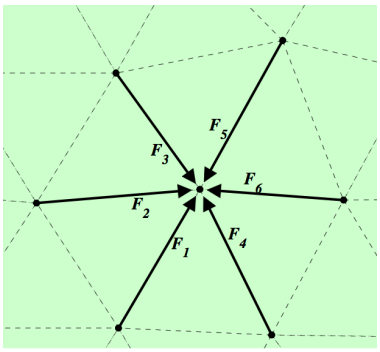


Figure 3.1: Force-based smoothing. The plot shows the net force exerted on one node. (Image from [61])

calculated using the following equation:

$$|\boldsymbol{F}(\boldsymbol{l}, \boldsymbol{l_0})| = \begin{cases} k(l_0 - l) & \text{if } l < l_0, \\ 0 & \text{if } l \geq l_0, \end{cases} \tag{3.1}$$

where k is a constant (corresponding to the spring constant in Hooke's law for a linear elastic spring).

The equilibrium length l_0 has to be set manually. For a uniform mesh, since the edge lengths are roughly the same, it is often set as a constant using the root mean square of all mesh edges:

$$l_0 = \sqrt{\frac{1}{n} \sum l_i^2}. \tag{3.2}$$

But for more general adaptive meshes l_0 has to be set individually for each edge, by a specified mesh size function $h(\vec{\boldsymbol{x}})$. Here we use the formula from [60], l_0 for the jth edge can be computed as:

$$l_{0,j} = h(\vec{\boldsymbol{x}}_j^{\text{mid}}) \sqrt{\frac{\sum l_i^2}{h(\vec{\boldsymbol{x}}_i^{\text{mid}})}}, \tag{3.3}$$

where $(\vec{\boldsymbol{x}}_i^{\text{mid}})$ are the coordinates of the mid-point of the ith edge. Note that if we choose $h(\vec{\boldsymbol{x}}) \equiv 1$ for the uniform mesh, the expression of Eq 3.3 is consistent with Eq 3.2.

In addition, a scaling factor slightly greater than 1 is often applied to l_0 for ensuring that most edges are under compression [60].

Based on our force formulation, we construct an artificial ODE system:

$$\frac{d\boldsymbol{p}}{dt} = \boldsymbol{F}(\boldsymbol{p}) \tag{3.4}$$

where $\boldsymbol{p}$ is the vector of node positions and $\boldsymbol{F}(\boldsymbol{p})$ is the sum of all forces from edges connected to each node. By a simple forward Euler scheme, we update the node position iteratively by

$$\boldsymbol{p}^{(n+1)} = \boldsymbol{p}^{(n)} + \delta \boldsymbol{F}(\boldsymbol{p}^{(n)}) \tag{3.5}$$

where δ is an appropriate pseudo time step. The iterations are repeated until an approximate force equilibrium is obtained (i.e. $\boldsymbol{F}(\boldsymbol{p}) \approx 0$ for every interior node).

Note that unlike the standard linear Hooke's law, Eq 3.1 zeroes out the attractive force when the edge length is larger than the equilibrium length, which is also a critical point to distinguish between the DistMesh approach and the popular Laplacian smoothing [25]. If we model the edge to behave like a regular spring according to Hooke's law, then the smoothing technique will move the node to the average position of all its neighbor nodes. On the contrary, without the attractive forces, DistMesh may move the nodes in a direction opposite to the average position. The main advantage of DistMesh is that we can have a mesh with better element qualities if we are allowed to change the mesh connectivity by some retriangulation algorithms (e.g. Delaunay triangulation [70]). Since we will change mesh connectivity frequently during the geometry motion with large deformations, the DistMesh approach is therefore preferred in our work.

Finally, it is straightforward to see that we can use the same force-based smoothing technique to update the positions of interior nodes when extending our moving mesh strategy to tetrahedral meshes in 3D.

3.2 Local Mesh Topology Change

When the time-dependent domain undergoes large deformations, node movements are usually not sufficient to obtain high-quality elements and avoid element inversion. For our mesh-moving strategy, however, we can perform local connectivity changes to improve the mesh qualities [2]. In the following, for each simplex element K of a triangulation $\mathcal{T}_t^h \in \mathbb{R}^d$, the mesh quality $Q(K)$ is defined by the measure proposed by Field [26]

$$Q(K) = d!(d)^{\frac{d}{2}}(d+1)^{\frac{d-1}{2}} \frac{\mathrm{Vol}(K)}{(\sum_{i=1}^{\frac{d(d+1)}{2}} l_i(K)^2)^{d/2}} \tag{3.6}$$

where $\mathrm{Vol}(K)$ is the volume of K and l_i is the length of each edge $i = 1, \ldots, \frac{d(d+1)}{2}$.

For a triangular mesh in 2D, this can be done using edge swapping operations [43] as shown in figure 3.2. Each time we consider a pair of adjacent triangles where usually at least one of them has an unsatisfactory element quality. We can then flip their shared edge and produce two new triangles sharing the new edge. If the minimum of the element qualities increases, we know that edge swapping does help improving the mesh and thus we accept this connectivity change. Note that during the process described, the number of nodes, edges and elements remain unchanged (we discuss density control for adding or removing mesh

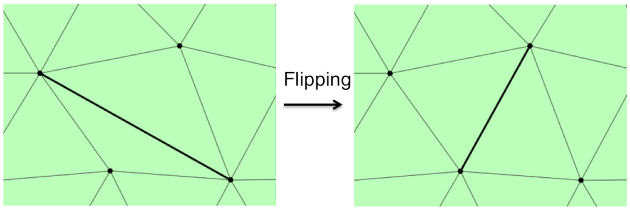


Figure 3.2: Local topology change in 2D.(Image from [61])

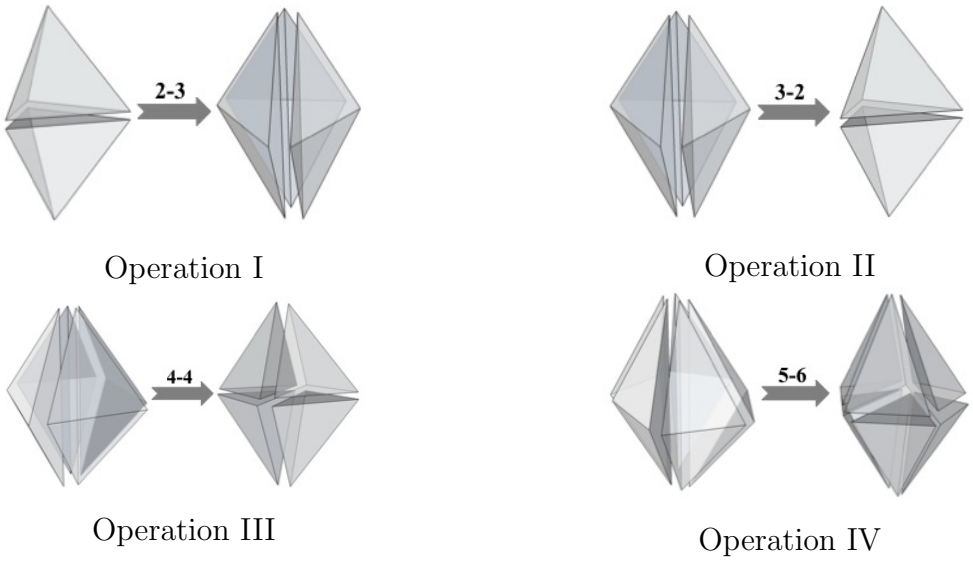


Figure 3.3: Change of local element connectivity in 3D. The numbers of old and new elements are shown above the arrows.

nodes below). Also, all elements have the same edge connections from $\mathcal{T}_t^h$ to $\mathcal{T}_{t+\Delta t}^h$ except for the ones involved in the edge swapping.

In the 3D case, we can change the mesh topology by the similar idea. However, the corresponding local connectivity changes need substantially more complicated operations in order to address element distortion in 3D. As shown in Fig. 3.3, we use 4 local connectivity change operations to improve the tetrahedra quality.

As the first operation, we consider a pair of tetrahedra sharing a common surface. These can be transformed into three new tetrahedra by removing the shared surface and introducing a new interior edge. The second one is the inverse operation to the first one, where we take three tetrahedra sharing a common edge and transform them into two tetrahedra sharing a common surface.

We use two more operations to deal with several tetrahedra at a time. When 4 tetrahedra share a common edge, then by removing that edge we can obtain a pair of adjacent pyramids.

To turn them into tetrahedra, we only have to assign a diagonal to their shared quadrilateral face. Similarly, for the case with 5 tetrahedra, the removal of common edge will give two polyhedra with a common pentagon. Two diagonals are then needed to triangulate the pentagon and hence triangulate the two polyhedra.

Based on our results, the operations above are sufficient to make the 3D mesh movement robust and keep the tetrahedra well-shaped. In principle, to make the mesh-moving strategy even more robust, we can follow the idea above and extend it to more cases [2]. Generally speaking, for any N tetrahedra with a shared edge, we can always transform them into $2(N-2)$ new tetrahedra by removing the common edge and triangulating the central polygon common face.

Comparing with regular remeshing or retriangulation algorithms, it turns out to be more efficient to use all the operations mentioned in this section. More importantly, when these operations are used, the connectivity change is local and limited to a small group of elements. This locality property builds the critical basis of our work and provides us the opportunity to simplify the space-time mesh generation as well as the solution transfer in the ALE framework, which will be discussed in details in chapter 4 and 5, respectively.

Note that in 3D, the number of mesh nodes remains the same before and after the mesh topology change but the number of elements and the edges might change during the process. This means the element sizes might change accordingly. Furthermore, for both 2D and 3D cases, there are also scenarios where we need to deliberately change the element sizes throughout the mesh motion. Next we will introduce our approaches to address this using so-called mesh adaptivity, using a similar locality property as above.

3.3 Mesh Adaptivity

In many applications with large domain deformations, a dynamically changing mesh size functions is needed as the mesh changes. Shock capturing problems are good examples [6], where in order to track the shock, we have to locally refine the mesh around the shock to better resolve the discontinuity and coarsen the mesh away from the shock to reduce the computational cost. Therefore, we can see that the mesh density control is an indispensable component for a mesh adjustment strategy. To incorporate this into our mesh-moving strategy, we present some additional local mesh operations in both 2D and 3D, in order to locally increase or decrease the number of elements when mesh density control is needed.

In 2D, as shown in figure 3.4, we split elements in two ways: First, this can be done within a single triangle, which is split into three smaller ones by adding an extra interior node connected to the other vertices; second, we can use the technique of edge splitting from [62], where the mesh is locally refined by adding the middle point of an edge and then splitting the two adjacent elements into four smaller ones. On the other hand, if a coarser mesh is needed, we consider two inverse cases to our refinement strategy: first, if a node is shared by three elements, we simply remove it and substitute by a new larger element; second, as an edge collapsing approach, if a node is shared by four elements, we remove it

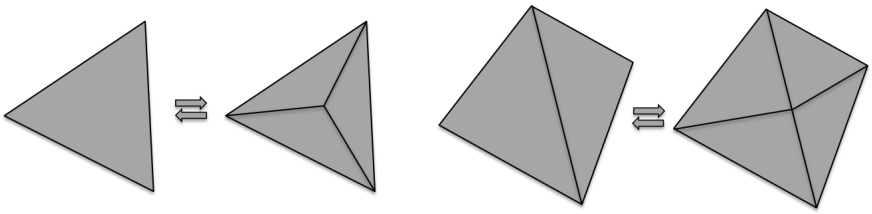


Figure 3.4: Density Control Operations in 2D. These operations are used to control the local mesh density by adding or removing nodes.

and locally re-triangulate the resulting quadrilateral into two triangular elements. Besides coarsening of the mesh, edge collapsing can also be used to improve the mesh quality by removing thin elements with big obtuse angles.

Again, although the 4 operations above turn out to perform well in our numerical test, in general, we can always involve more density control operations with respect to a bigger group of triangles (e.g. 5 or 6 triangles sharing a common vertex, which we can remove and retriangulate the outside pentagon or hexagon). Moreover, note that we locally adjust the mesh density by operating on triangles and quadrilaterals, which is consistent with the elements with or without edge swapping from section 3.2, respectively. For our space-time mesh generation which will be introduced in chapter 4, this is a an important premise in order to unify the local triangulation analysis for elements involved in edge swapping and in density changes.

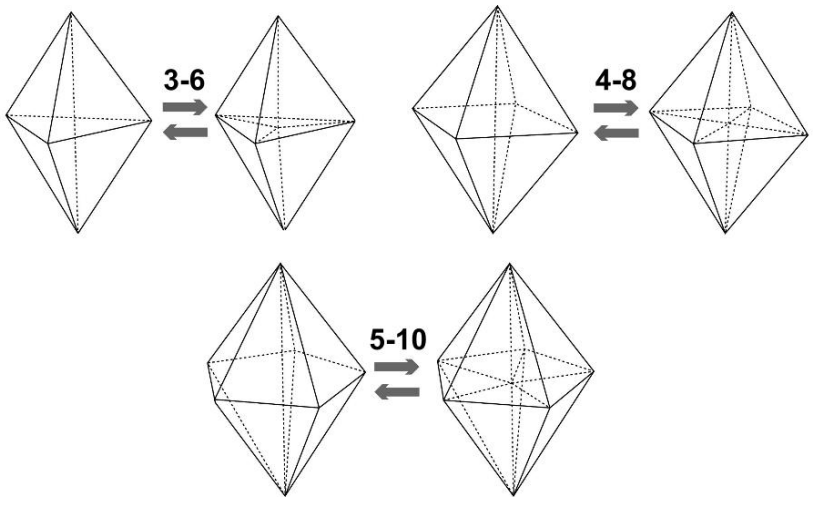


Figure 3.5: Density Control Operations in 3D. These operations are used to control the local mesh density by adding or removing nodes.

Similarly, we propose the density control operations associated with our local connectivity change operations in 3D. Again, as shown in figure 3.5, consider the cases when multiple tetrahedra share a common edge. By the idea of edge splitting, we can pick up the middle point of this common edge and connect all the other vertices to it. In this way, we can split N tetrahedra and obtain $2N$ new elements with smaller volume. We choose $N = 3, 4, 5$, consistent with our 3D topology change operations. Inversely, we recover the original coarse mesh by removing the middle point and the added edges.

3.4 The algorithm

Based on the operations in section 3.1, 3.2 and 3.3, we present the complete algorithm for our moving-mesh strategy in algorithm 3.1.

Algorithm 3.1 Moving-Mesh Startegy

Require: Triangulation $\mathcal{T}_0^h$ on Ω^0
Require: Time step Δt, time-dependent mesh size function $h(\vec{\boldsymbol{x}}, t)$, number of sweeps for node smoothing N, pseudo time-step δ and mesh quality threshold ξ
Ensure: Triangulation $\mathcal{T}_{t_i}^h$ for each time step t_i until time T
  Set $t_i = 0$
  while $0 \leq t_i < T$ **do**
    Move the boundary nodes rigidly according to geometry motion.
    Set $\boldsymbol{p}^{(0)}$ as the current position of each interior node.
    for i from 0 to N-1 **do**
      Compute the net force $\boldsymbol{F}(\boldsymbol{p}^{(i)})$ for each $\boldsymbol{p}^{(i)}$.
      Update node position by $\boldsymbol{p}^{(i+1)} = \boldsymbol{p}^{(i)} + \delta \boldsymbol{F}(\boldsymbol{p}^{(i)})$
    end for
    Set $\boldsymbol{p}^{(N)}$ as the current position of each interior node.
    while some element sizes are inconsistent with the ratio implied by $h(\vec{\boldsymbol{x}}, t)$ **do**
      Change the mesh adaptivity using operations in section 3.3.
    end while
    while min quality of element $K \in \mathcal{T}_{t_i}^h < \xi$ **do**
      Change the local mesh connectivity with operation from section 3.2
    end while
    Update $t_i = t_i + \Delta t$
  end while

At each time step, first the spring-based node smoothing will improve the mesh quality by moving the nodes and changing the edge lengths. In practice, the absolute equilibrium state does not have to be reached and we usually smooth the nodes with a fixed number of sweeps (i.e. the input N in algorithm 3.1). The pseudo time-step δ is often chosen smaller than Δt, such that the typical value of number of sweeps N is about 10-20 per Δt.

Next, given a time-dependent mesh size function $h(\vec{\boldsymbol{x}}, t)$, we change the mesh density if needed. When applying our local density control operations, we employ a greedy approach and start changing local density of the elements whose size ratio compared to the desired value $h(\vec{\boldsymbol{x}}, t)$ is the largest. We attempt the local density control operations using the operations with the most elements involved. For example, in 2D, the edge splitting approach has the higher priority. When all the neighbors of a triangle have their desired element size, the single splitting approach will be applied. Since we know that the greedy algorithm often finds a sub-optimal solution, our method can adjust the density for most regions but may not guarantee that every element satisfies the requirement from $h(\vec{\boldsymbol{x}}, t)$. On the other hand, some new elements may not have sufficiently good quality after the mesh refinement especially for the single element splitting. For those concerns, the operations from section 3.2 are implemented at the end of each loop to locally change mesh topology and further enhance the element quality. Similar to the density control stage, we use a greedy algorithm and start to change the mesh connectivity for the elements with worst quality.

Note that a while loop is used for both the density control and the connectivity update parts, indicating that we are able to carry out the greedy algorithm for multiple sweeps. We require that during each sweep, each element can be involved in at most one operation to reinforce the locality property. This requirement plays a vital role when doing local L^2 projections in the ALE framework in chapter 5. For space-time methods, during each time step, we only allow either one sweep of density control or connectivity update in order to generate the space-time meshes. In chapter 4, we will discuss ways to make this requirement less restrictive.

Using the algorithm above, we can retain the mesh topology for all elements except for the ones involved in the local operations. In the next chapters, we will talk about how to take advantage of these properties and propose efficient methods based on them.

Chapter 4

Space-Time Discontinuous Galerkin Methods

Discretization methods based on the so-called space-time framework unify the spatial and the temporal dimensions, and solve the resulting system of equations without a separate time integrator. In this chapter, we present a space-time discontinuous Galerkin method. It consists of two major parts: first we give the complete numerical scheme including the discretization and the solver, next we describe our space-time mesh generator based on the spatial meshes created using the techniques in chapter 3.

4.1 Space-Time Formulation

Recall that we introduced the conservation form of the compressible Navier-Stokes equations in Eq 2.1. Here we use the general expression from Eq 2.7:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F}^{\text{inv}}(\boldsymbol{u}) = \nabla \cdot \boldsymbol{F}^{\text{vis}}(\boldsymbol{u}, \nabla \boldsymbol{u}), \tag{4.1}$$

Consider these equations on a time-varying domain in $\mathbb{R}^n$ between time $t = 0$ to $t = T$ for some fixed final time $T > 0$. Let $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ be the spatial variables. Again, we denote $\Omega^t \subset \mathbb{R}^d$ as this domain at time t and when $t_1 < t_2$, we define the corresponding space-time domain $\Omega[t_1, t_2] = \{(\mathbf{x}, t) \mid t_1 \leq t \leq t_2, \ \ \mathbf{x} \in \Omega^t\}$. Next, by treating the temporal dimension as an additional 'spatial' dimension. we transform Eq 4.1 from the d-dimensional spatial domain Ω^t between times $t = 0$ and $t = T$, into our space-time formulation of equations in the $d + 1$-dimensional space-time domain $\Omega[0, T]$. We introduce a new space-time gradient operator $\nabla_{\boldsymbol{st}} = (\partial_{x_1}, \partial_{x_2}, \ldots, \partial_{x_d}, \partial_t)$ and rewrite the Navier-Stokes equations in $\Omega[0, T]$ as

$$\nabla_{\boldsymbol{st}} \cdot \tilde{\boldsymbol{F}}^{\text{inv}}(\boldsymbol{u}) = \nabla \cdot \boldsymbol{F}^{\text{vis}}(\boldsymbol{u}, \nabla \boldsymbol{u}), \tag{4.2}$$

where the new inviscid space-time fluxes $\tilde{\boldsymbol{F}}^{\text{inv}}(\boldsymbol{u})$ have $d+1$ components. By attaching the vector solution $\boldsymbol{u}$ to the end of the regular fluxes $\tilde{\boldsymbol{F}}^{\text{inv}}(\boldsymbol{u})$, it can be expressed as

$$\tilde{\boldsymbol{F}}^{\text{inv}}(\boldsymbol{u}) = (\tilde{\boldsymbol{F}}_1^{\text{inv}}(\boldsymbol{u}), \tilde{\boldsymbol{F}}_2^{\text{inv}}(\boldsymbol{u}), \ldots, \tilde{\boldsymbol{F}}_{d+1}^{\text{inv}}(\boldsymbol{u})) = (\boldsymbol{F}^{\text{inv}}(\boldsymbol{u}), \boldsymbol{u}) \tag{4.3}$$

Note that since there are no temporal derivatives involved in the viscous fluxes, we retain the same term $\boldsymbol{F}^{\text{vis}}(\boldsymbol{u}, \nabla \boldsymbol{u})$ on the right hand side of Eq 4.2.

Finally, the boundary conditions on $\partial\Omega[0,T]$ for Eqs 4.2 are the same as the boundary conditions given for the original Eqs 4.1 (we refer the details to section 2.3). The boundary conditions on the bottom face Ω^0 correspond to the given initial conditions of Eqs 4.1. On the top boundary Ω^T of $\Omega[0,T]$, no boundary conditions are needed for the space-time formulation, since the temporal derivative is treated as a simple linear convection term and the corresponding characteristics move in the positive direction along the temporal direction.

4.2 Numerical Scheme

4.2.1 Discretization of the Euler Equations

First, we describe the nodal discontinuous Galerkin discretization for the inviscid part. Consider the first-order space-time formulation of the Euler equations,

$$\nabla_{\boldsymbol{st}} \cdot \tilde{\boldsymbol{F}}^{\text{inv}}(\boldsymbol{u}) = \boldsymbol{0}. \tag{4.4}$$

We introduce a conforming triangulation $\mathcal{T}^h_{[0,T]} = \{K\}$ of the d-dimensional space-time domain $\Omega[0,T]$ into tetrahedral elements K. On $\mathcal{T}^h_{[0,T]}$, we define the broken space $\mathcal{V}^h_{[0,T]}$ as the space of functions whose restriction to each element K are polynomial functions of degree at most $p \geq 1$ [31]:

$$\mathcal{V}^h_{[0,T]} = \{\boldsymbol{v} \in [L^2(\Omega[0,T])]^{d+2} \mid \boldsymbol{v}|_K \in [\mathcal{P}_p(K)]^{d+2} \ \ \forall K \in \mathcal{T}^h_{[0,T]}\}, \tag{4.5}$$

where $\mathcal{P}_p(K)$ denotes the space of polynomials of degree at most $p \geq 1$ on K.

The nodal DG method follows the standard finite element approach and chooses a set of $N_p = \binom{p+d}{d}$ equidistributed nodes $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{N_p}\}$ within each element K in the d-dimensional space (p is again the polynomial degrees). Unlike the continuous Galerkin formulation, the solution is locally represented by a linear combination of N_p shape functions $\{\phi_1^h, \phi_2^h, \ldots, \phi_{N_p}^h\}$ within each element K, where each ϕ_i^h is defined as the Lagrange interpolation functions with respect to each $\boldsymbol{x}_i$ (i.e. $\phi_i^h \in \mathcal{P}_p(K)$ and $\phi_i^h(\boldsymbol{x}_j) = \delta_{ij}$). Therefore, as shown in figure 4.1, we can see that along each interior face, the nodes are duplicated between neighboring elements. This locality property creates discontinuities between solutions from different elements, which provides an opportunity to define numerical fluxes similar to the finite volume method in order to stabilize the solution[44].

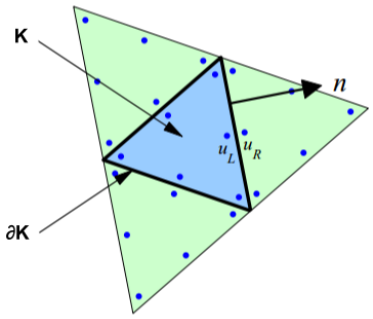


Figure 4.1: An example for node distribution of DG method. Here, $p = 2$ and 6 nodes are chosen within each element K. (Image from [61])

Accordingly, for each $K \in \mathcal{T}^h_{[0,T]}$, the solution $\boldsymbol{u}^h \in \mathcal{V}^h_{[0,T]}$ can be written as

$$\boldsymbol{u}^h = \sum_{k=1}^{N_p} \boldsymbol{u}_k \phi^h_k(\boldsymbol{x}) \tag{4.6}$$

where $\boldsymbol{u}_k \in \mathbb{R}^{d+2}$ is a coefficient vector which we are solving for. Based on this expression, our DG formulation for Eq 4.4 then becomes: find $\boldsymbol{u}^h \in \mathcal{V}^h_{[0,T]}$ such that

$$\begin{aligned}\int_K \left(\nabla_{\boldsymbol{st}} \cdot \tilde{\boldsymbol{F}}^{\text{inv}}(\boldsymbol{u}^h)\right) \cdot \boldsymbol{v}^h \, dx &= -\int_K \tilde{\boldsymbol{F}}^{\text{inv}}(\boldsymbol{u}^h) : \nabla_{\boldsymbol{st}} \boldsymbol{v}^h \, dx + \oint_{\partial K} (\widehat{\tilde{\boldsymbol{F}}^{\text{inv}} \cdot \boldsymbol{n}}) \cdot \boldsymbol{v}^h \, ds \\ &= 0, \qquad \forall \boldsymbol{v}^h \in \mathcal{V}^h_{[0,T]}.\end{aligned} \tag{4.7}$$

Here, $\boldsymbol{v}^h$ are test functions and $\boldsymbol{n} \in \mathbb{R}^{d+1}$ is the outward unit normal to the boundary ∂K of the space-time tetrahedron K.

We define the spatial part of the normal vector as $\boldsymbol{n}_s$ and the temporal part as $\boldsymbol{n}_t$, and similarly define the spatial part of the flux term as $\tilde{\boldsymbol{F}}^{\text{inv}}_s$ and the temporal part as $\tilde{\boldsymbol{F}}^{\text{inv}}_t$. If we normalize the spatial and the temporal components of the normal vector, respectively, as $\tilde{\boldsymbol{n}}_s = \boldsymbol{n}_s/|\boldsymbol{n}_s|$ and $\tilde{n}_t = n_t/|n_t|$, we can decompose the numerical flux into two parts as

$$\widehat{\tilde{\boldsymbol{F}}^{\text{inv}} \cdot \boldsymbol{n}} = |\boldsymbol{n}_s| \left[\widehat{\tilde{\boldsymbol{F}}^{\text{inv}}_s \cdot \tilde{\boldsymbol{n}}_s}\right] + |n_t| \left[\widehat{\tilde{\boldsymbol{F}}^{\text{inv}}_t \tilde{n}_t}\right] = |\boldsymbol{n}_s| \boldsymbol{\mathcal{F}}^s + |n_t| \boldsymbol{\mathcal{F}}^t. \tag{4.8}$$

The numerical flux $\widehat{\tilde{\boldsymbol{F}}^{\text{inv}} \cdot \boldsymbol{n}}$ is an approximation to $\tilde{\boldsymbol{F}}^{\text{inv}} \cdot \boldsymbol{n}$ on the face of element K, which is specified in terms of $\boldsymbol{u}^h$ on the two sides of the element boundary and by the boundary conditions.

For the first term of $\widehat{\tilde{\boldsymbol{F}}^{\text{inv}} \cdot \boldsymbol{n}}$, we define the spatial numerical flux $\boldsymbol{\mathcal{F}}^s$ as the standard approximate Riemann solver proposed by Roe [68]. Roe's method linearizes the nonlinear

fluxes $\tilde{\boldsymbol{F}}_s^{\text{inv}}(\boldsymbol{u}) \cdot \tilde{\boldsymbol{n}}_s$ by introducing Roe's matrices. More precisely, we denote $\boldsymbol{u}^+$ as the solution exterior to element K and the $\boldsymbol{u}^-$ as that interior to K. Then the flux can be approximated by:

$$\tilde{\boldsymbol{F}}_s^{\text{inv}}(\boldsymbol{u}) \cdot \tilde{\boldsymbol{n}}_s \approx \tilde{A}\boldsymbol{u} \tag{4.9}$$

where $\tilde{A} = \tilde{A}(\boldsymbol{u}^+, \boldsymbol{u}^-)$ are Roe's matrices constructed by a linear parameterization of $\boldsymbol{u}$ and $\tilde{\boldsymbol{F}}_s^{\text{inv}}$, which satisfy:

- $\tilde{A}$ is diagonalizable with real eigenvalues
- When $\boldsymbol{u}^+ \to \boldsymbol{u}$ and $\boldsymbol{u}^- \to \boldsymbol{u}$, $\tilde{A}$ converges to the exact Jacobian matrix $\frac{\partial(\tilde{\boldsymbol{F}}_s^{\text{inv}}(\boldsymbol{u}) \cdot \tilde{\boldsymbol{n}}_s)}{\partial \boldsymbol{u}}$
- For any $\boldsymbol{u}^+$, $\boldsymbol{u}^-$, $\tilde{A}(\boldsymbol{u}^+ - \boldsymbol{u}^-) = (\tilde{\boldsymbol{F}}_s^{\text{inv}}(\boldsymbol{u}^+) - \tilde{\boldsymbol{F}}_s^{\text{inv}}(\boldsymbol{u}^-)) \cdot \tilde{\boldsymbol{n}}_s$

Then by a standard Riemann solver approach, $\boldsymbol{\mathcal{F}}^s$ can be computed as

$$\boldsymbol{\mathcal{F}}^s = \{\tilde{\boldsymbol{F}}_s^{\text{inv}} \cdot \tilde{\boldsymbol{n}}_s\} - \frac{1}{2}|\tilde{A}|[\boldsymbol{u} \otimes \tilde{\boldsymbol{n}}_s] \tag{4.10}$$

with the eigen-decomposition $\tilde{A} = R\Lambda R^{-1}$, $|\tilde{A}|$ is defined as $R|\Lambda|R^{-1}$. $\{\cdot\}$ and $[\cdot]$ are the arithmetic mean and difference between interior (terms with 'minus' sign) and exterior (terms with 'plus' sign) quantities along the element interface. Their definitions are as follows:

$$\begin{aligned} \{\boldsymbol{v}\} &= (\boldsymbol{v}^+ + \boldsymbol{v}^-)/2 \qquad [\boldsymbol{v}] = \boldsymbol{v}^+ \otimes \boldsymbol{n}^+ + \boldsymbol{v}^- \otimes \boldsymbol{n}^- \qquad \forall \boldsymbol{v} \in \mathbb{R}^m \\ \{\boldsymbol{v}\} &= (\boldsymbol{v}^+ + \boldsymbol{v}^-)/2, \qquad [\boldsymbol{v}] = \boldsymbol{v}^+ \cdot \boldsymbol{n}^+ + \boldsymbol{v}^- \cdot \boldsymbol{n}^- \qquad \forall \boldsymbol{v} \in \mathbb{R}^{m \times d} \end{aligned} \tag{4.11}$$

where m is the number of componenets and d is the dimension. For more details, we refer to [68].

For the second term on the right hand side of Eq 4.8, we define the temporal numerical flux $\boldsymbol{\mathcal{F}}^t$ by the standard upwinded flux of the corresponding linear time-derivative term $\boldsymbol{u}_t$ in Eq 4.4:

$$\boldsymbol{\mathcal{F}}^t = \begin{cases} \boldsymbol{u}^+ \tilde{n}_t & \text{if } \tilde{n}_t > 0, \\ \boldsymbol{u}^- \tilde{n}_t & \text{otherwise.} \end{cases} \tag{4.12}$$

Note that on the boundaries Ω^0 and Ω^T, the boundary conditions are indirectly incorporated by the temporal numerical fluxes $\boldsymbol{\mathcal{F}}^{\boldsymbol{t}}$. In particular, since these are defined by upwinding, the initial conditions of equations (2.1) are used on Ω^0 and the interior solutions on Ω^T. This property makes it possible to advance the solution for a single interval Δt at a time, without connecting the entire space-time solution domain. In this sense, the space-time DG formulation is similar to a standard implicit method-of-lines formulation using a one-step time integrator.

The discretization above results in a final non-linear algebraic system of equations without any explicit time dependencies. Note that unlike method-of-lines discretizations, the time discretization is part of the DG formulation and thus it does not allow for choosing other methods, such as multistep or explicit time integrators. We will discuss numerical solvers for Eq 4.26 in section 4.2.3.

4.2.2 Discretization of the Viscosity Terms

Next we describe the discretization of the viscous terms in the compressible Navier-Stokes equations. We apply a standard procedure for second-order terms [4], where the system (4.2) is split into a first-order system of equations by introducing new unknown variables $\boldsymbol{q}$:

$$\nabla_{\boldsymbol{st}} \cdot \tilde{\boldsymbol{F}}^{\text{inv}}(\boldsymbol{u}) = \nabla \cdot \boldsymbol{F}^{\text{vis}}(\boldsymbol{u}, \boldsymbol{q}), \tag{4.13}$$

$$\nabla \boldsymbol{u} = \boldsymbol{q}. \tag{4.14}$$

This system is again discretized using a standard DG procedure. First, we introduce an additional broken space $\Sigma^h_{[0,T]}$ for the approximation $\boldsymbol{q}^h$ as

$$\Sigma^h_{[0,T]} = \{\boldsymbol{\sigma} \in [L^2(\Omega[0,T])]^{(d+2)\times d} \mid \boldsymbol{\sigma}|_K \in [\mathcal{P}_p(K)]^{(d+2)\times d} \ \ \forall K \in \mathcal{T}^h_{[0,T]}\}, \tag{4.15}$$

and $\boldsymbol{q}^h$ also follows the Galerkin expression as Eq 4.6

$$\boldsymbol{q}^h = \sum_{k=1}^{N_p} \boldsymbol{q}_k \phi^h_k(\boldsymbol{x}) \tag{4.16}$$

where $\boldsymbol{q}_k \in \mathbb{R}^{(d+2)\times d}$ is a coefficient matrix which we are solving for.

Then the DG formulation for equations (4.13) and (4.14) becomes: find $\boldsymbol{u}^h \in \mathcal{V}^h_{[0,T]}$ and $\boldsymbol{q}^h \in \Sigma^h_{[0,T]}$ such that for each $K \in \mathcal{T}^h_{[0,T]}$, we have

$$\begin{aligned} -\int_K \tilde{\boldsymbol{F}}^{\text{inv}}(\boldsymbol{u}^h) : \nabla_{\boldsymbol{st}} \boldsymbol{v}^h \, dx + \oint_{\partial K} (\widehat{\tilde{\boldsymbol{F}}^{\text{inv}} \cdot \boldsymbol{n}}) \cdot \boldsymbol{v}^h \, ds \\ = -\int_K \boldsymbol{F}^{\text{vis}}(\boldsymbol{u}^h, \boldsymbol{q}^h) : \nabla \boldsymbol{v}^h \, dx + \oint_{\partial K} (\widehat{\boldsymbol{F}^{\text{vis}} \cdot \boldsymbol{n}_s}) \cdot \boldsymbol{v}^h \, ds, \quad \forall \boldsymbol{v}^h \in \mathcal{V}^h_{[0,T]} \end{aligned} \tag{4.17}$$

$$\int_K \boldsymbol{q}^h : \boldsymbol{\sigma}^h \, dx = -\int_K \boldsymbol{u}^h \cdot (\nabla \cdot \boldsymbol{\sigma}^h) \, dx + \oint_{\partial K} (\widehat{\boldsymbol{u}}^h \otimes \boldsymbol{n}_s) \cdot \boldsymbol{\sigma}^h \, ds, \quad \forall \boldsymbol{\sigma}^h \in \Sigma^h_{[0,T]}. \tag{4.18}$$

Recall that $\boldsymbol{n_s}$ is the spatial component of the outward unit normal $\boldsymbol{n}$ at the boundary ∂K.

In Eqs 4.17 and 4.18, the inviscid part $\widehat{\tilde{\boldsymbol{F}}^{\text{inv}} \cdot \boldsymbol{n}}$ is approximated using the same procedures as previously described in section 4.2.1. On the other hand, the viscous fluxes $\widehat{\boldsymbol{F}^{\text{vis}} \cdot \boldsymbol{n}_s}$ and $\widehat{\boldsymbol{u}^h}$ are approximations to $\boldsymbol{F}^{\text{vis}} \cdot \boldsymbol{n}_s$ and $\boldsymbol{u}$ on the boundary of element K. For the viscous numerical flux, we first perform a similar normalization:

$$\widehat{\boldsymbol{F}^{\text{vis}} \cdot \boldsymbol{n}}_s = |\boldsymbol{n}_s| \widehat{\boldsymbol{F}^{\text{vis}} \cdot \tilde{\boldsymbol{n}}}_s. \tag{4.19}$$

Note that since the viscous terms in Eqs 4.13 as well as all of the equations 4.14 only involve derivatives with respect to the spatial variables, we can approximate $\widehat{\boldsymbol{F}^{\text{vis}} \cdot \tilde{\boldsymbol{n}}_s}$ and $\widehat{\boldsymbol{u}^h}$ using schemes for spatial discretizations without modifications.

There are many approaches designed for discretizing the second order terms in the compressible Navier-Stokes Equations and a survey of various methods can be found in [10].

Some direct discretization techniques include the symmetric interior penalty method (IP) [22], the BR2 method [7], and the local discontinuous Galerkin (LDG) scheme [13]. Here, we choose the numerical fluxes according to the Compact Discontinuous Galerkin (CDG) method [54], which is a modified version of the LDG method with a more compact sparsity pattern and better stability properties.

Using the notation from Eq 4.11, the CDG method defines $\widehat{\boldsymbol{F}^{\text{vis}} \cdot \tilde{\boldsymbol{n}}_s} = \widehat{\boldsymbol{F}^{\text{vis}}} \cdot \tilde{\boldsymbol{n}}_s$ and $\widehat{\boldsymbol{u}^h}$ in the form of

$$\widehat{\boldsymbol{F}^{\text{vis}}} = \{\boldsymbol{F}^{\text{vis}}\} + C_{11}[\boldsymbol{u}] + [\boldsymbol{F}^{\text{vis}}] \otimes \boldsymbol{C}_{12} \tag{4.20}$$

$$\widehat{\boldsymbol{u}^h} = \{\boldsymbol{u}^h\} - \boldsymbol{C}_{12} \cdot [\boldsymbol{u}^h] + C_{22}[\boldsymbol{F}^{\text{vis}}] \tag{4.21}$$

where the coefficients C_{11} and C_{22} are scalar and $\boldsymbol{C}_{12}$ is a vector in the direction of the normal vector. Different choices of these coefficients induce different schemes. The CDG method relies on a set of switch functions on each internal face of the elements. More precisely, assume element K_1 and element K_2 share a common face. Then on the side of K_1, we define a switch function S_{K_1,K_2} and S_{K_2,K_1} on the side of K_2 (i.e. each face has two switch functions for two sides). Each switch function has two possible values chosen from the set $\{-1, 1\}$ and satisfies the condition:

$$S_{K_2,K_1} + S_{K_1,K_2} = 0 \tag{4.22}$$

There are various ways to assign the values of these switch functions. A simple approach is the natural switch, where we globally index all the element, and assign $S_{K_1,K_2} = 1$ if the index of K_1 is greater than that of K_2 and $S_{K_1,K_2} = -1$ otherwise.

Consider an element K_1 with a neighbor K_2. Based on the value of switch functions, the coefficient $\boldsymbol{C}_{12}$ of K_1 associated with the shared face is determined by

$$\boldsymbol{C}_{12} = \frac{1}{2}(S^{K_1,K_2}\tilde{\boldsymbol{n}}_s^+ + S^{K_2,K_1}\tilde{\boldsymbol{n}}_s^-) \tag{4.23}$$

Then the numerical fluxes in Eqs 4.20 and 4.21 are as follows:

$$\widehat{\boldsymbol{F}^{\text{vis}}} = C_{11}[\boldsymbol{u}] + \begin{cases} \boldsymbol{F}^{\text{vis}}(\boldsymbol{u}^+, \boldsymbol{q}^+), & \text{if } S^{K_1,K_2} = 1, \\ \boldsymbol{F}^{\text{vis}}(\boldsymbol{u}^-, \boldsymbol{q}^-), & \text{if } S^{K_1,K_2} = -1, \end{cases} \tag{4.24}$$

$$\widehat{\boldsymbol{u}^h} = C_{22}[\boldsymbol{F}^{\text{vis}}] + \begin{cases} \boldsymbol{u}^-, & \text{if } S^{K_1,K_2} = 1, \\ \boldsymbol{u}^+, & \text{if } S^{K_1,K_2} = -1. \end{cases} \tag{4.25}$$

Through Eqs 4.24 and 4.25, we can see that numerical fluxes from the CDG method have an upwinding/downwinding character related to the value of the switch functions. Besides, the first term on the right hand side of each equation can be regarded as a penalty terms for additional stability. In our implementation, we usually set the constants C_{11} and C_{22} to zero. Note that by this setting, the dependence of $\boldsymbol{q}$ is eliminated from Eq 4.25 which means all the auxiliary $\boldsymbol{q}$ variables can be eliminated from the system using local operations.

Eqs 4.17 and 4.18 define a non-linear discrete system for $\boldsymbol{u}$ and $\boldsymbol{q}$, and after elimination of $\boldsymbol{q}$ the final system only involves $\boldsymbol{u}$. Next, we will describe our numerical solver for this non-linear system.

4.2.3 Newton-Krylov Solver

As described in sections 4.2.1 and 4.2.1, the DG discretization for the space-time formulation finally results in a time-independent nonlinear system of the form

$$\boldsymbol{R}(\boldsymbol{u}^h) = 0. \tag{4.26}$$

To solve this system, we use Newton's method and the efficient parallel block-ILU(0) preconditioned restarted GMRES method proposed in [58]. Newton's method is a standard choice for solving non-linear algebraic system. For this particular problem, denote $\boldsymbol{u}^{h(k)}$ as the approximation solution after k iterations. At each iteration, we compute the correction term δu and update our solution by the following scheme

$$\boldsymbol{J}\delta u = \boldsymbol{R}(\boldsymbol{u}^{h(k)}), \tag{4.27}$$

$$\boldsymbol{u}^{h(k+1)} = \boldsymbol{u}^{h(k)} - \delta u \tag{4.28}$$

where $\boldsymbol{J}$ is the Jacobian matrix $\frac{d\boldsymbol{R}}{d\boldsymbol{u}}$.

Due to the large size of the linear system 4.27 and the relatively dense structure of the Jacobian matrices, it is quite difficult to invert the Jacobian matrix using direct sparse factorizations. Instead, there are various Krylov subspace methods to iteratively solver for the solution such as the quasi-minimal residual method (QMR), the conjugate gradient squared method (CGS) and the generalized minimal residual method (GMRES). A detailed discussion about Krylov method can be found in [18]. In our work, we choose GMRES for Eq 4.27.

In addition, it is well known that the performance of GMRES can be significantly improved by an appropriate preconditioner. Note that the CDG method has a compact property such that for each element K, Eqs 4.17 and 4.18 only involve the degrees of freedom in K and its neighboring elements. This gives $\boldsymbol{J}$ a block-wise structure, where each row of $\boldsymbol{J}$ corresponding to a node in K only has non-zero entries in the blocks corresponding to K and its neighbors. Based on this special structure of the Jacobian matrices, we can use a block-ILU(0) preconditioner with our GMRES solver. In short, this preconditioner computes an incomplete LU factorization for the Jacobian matrix $\boldsymbol{J}$ by applying standard Gaussian elimination on each non-zero block, and ensures that no non-zero entry can be added outside the sparsity pattern of $\boldsymbol{J}$.

Finally, since high-order discretizations always introduce large-scale computation and storage costs, it is necessary to implement our numerical solver in parallel to improve the performance. Here we refer to [56] for the details of our partitioning strategy.

4.3 Space-Time Mesh Generation for 2D Problems

From the description of the DG discretization above, we can see that when we solve moving domain problems with large deformations, one key benefits of the space-time framework is that it does not require any explicit time integration procedure. For the traditional method-of-lines approach, the time integration must be computed along the time characteristics, which cannot be directly used when the spatial mesh topology changes. Instead, the space-time framework depends on a unified space-time mesh and thus has an unstructured mesh pattern also in the temporal dimension. This property provides much more flexibility to deal with large deformation problems. In this section, we present our novel algorithm for space-time mesh generation for 2D problems.

For 2D problems, the space-time mesh needs to be created in 3D. Most of the previous literature use commercial 3D tetrahedral mesh generators with constraints on the surface mesh (i.e. the surface mesh must be consistent with the spatial meshes). However, this approach may introduce more elements than necessary, which will significantly increase the computational costs for our fully implicit numerical solver. Instead, our approach is closely related to the moving-mesh techniques in section 3. We try to take advantage of the relationship among our spatial meshes and design a more efficient mesh generator.

4.3.1 Basic Idea

Given a timestep Δt, in order to reduce the computational cost, we generate tetrahedral space-time meshes for each slab $\Omega[t, t + \Delta t]$ separately (see figure 4.2). Since our moving-mesh algorithm is entirely based on local mesh modifications, the space-time mesh can be both efficiently and robustly generated directly from these operations.

More specifically, given an unstructured mesh of $\Omega^t \in \mathbb{R}^2$ at time t, we first generate an unstructured mesh of $\Omega^{t+\Delta t}$ as the time-dependent flow domain is deforming, using our moving-mesh techniques. Based on the resulting two layers of triangular meshes, we apply an efficient combinatorial tetrahedral triangulation method to generate the space-time mesh of $\Omega[t, t + \Delta t]$. We then solve the compressible Navier-Stokes equations in this space-time mesh using the DG scheme described in the previous section, and repeat the procedure for the next space-time slab $\Omega[t + \Delta t, t + 2\Delta t]$, etc. The domains Ω^t are never re-meshed from scratch, instead only one initial mesh generation for Ω^0 is needed, which is then improved at each subsequent time step. More importantly, all the mesh improvement techniques are performed on the 2D spatial mesh, and the tetrahedral triangulation is entirely based on local combinatorial connections.

Recall that our spatial mesh generator is based on the DistMesh algorithm [60], which iteratively improves a triangular mesh using only node movements, element connectivity updates and density control. As we mentioned in section 3.4, to simplify the space-time tetrahedral triangulation algorithm, we require that each element is only involved in at most one sweep of density control or connectivity change during each time step. This means that in 2D, all the elements involving topology changes come in groups. This simplification does

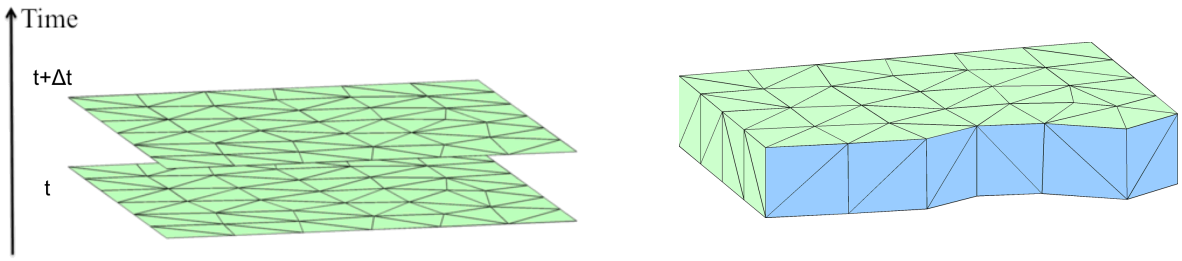


Figure 4.2: Space-Time Mesh Generation. The left figure illustrates two mesh layers at time t and $t + \Delta t$, and the right figure shows a corresponding 3D space-time mesh between the two layers. The blue faces show a cross-section of the tetrahedral mesh.

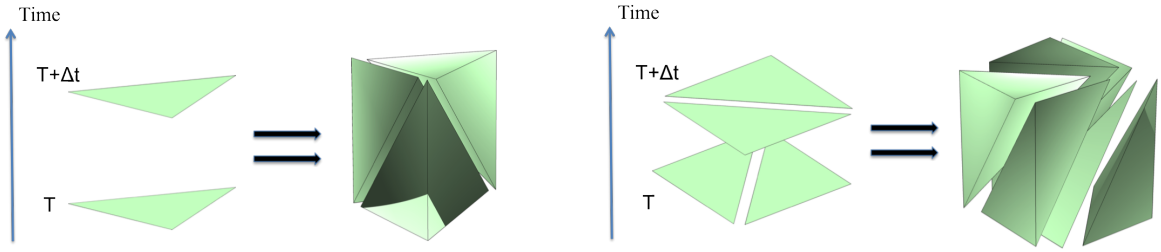


Figure 4.3: Tetrahedral Triangulation in 2D. The left plot illustrates a valid triangulation for an element without edge flips, and the right plot shows a triangulation for a pair of elements with a flipped edge.

not appear to impose a severe limitation in any of our numerical tests, which include highly complex domain motions and deformations. In principle it is possible that multiple sweeps may be required to obtain high element qualities, in which case we simply reduce the time step. Another effective way to further alleviate this limitation is to restrict the mesh topology changes to certain regions of the domain and keep other parts of the mesh either fixed or rigidly following the boundary motion. By performing local mesh adaptation only on the regions which allow for mesh topology changes, we obtain a sufficient number of groups of elements and the local operations are able to generate meshes with high quality.

The next step is to efficiently generate a space-time mesh $\Omega[t, t + \Delta t]$ for each time step based on the initial mesh of the spatial domain Ω^t and the deformed and improved mesh of $\Omega^{t+\Delta t}$. We begin with considering the case without density control operations. Recall that our 2D mesh moving and edge flipping algorithm is able to keep the same number of nodes on $\Omega^{t+\Delta t}$ as that of Ω^t, so we can simply connect each node of Ω^t with its corresponding node of $\Omega^{t+\Delta t}$, as the first step of our space-time mesh generation. This point-wise connection will ensure that the space-time mesh respects the moving boundary, due to the rigid motion of boundary nodes from Ω^t to $\Omega^{t+\Delta t}$.

First, consider an element of Ω^t without edge flipping (figure 4.3, left). This element can be extruded to $\Omega^{t+\Delta t}$ and form an irregular triangular prism, where 'irregular' means that the

edge on the bottom face is not necessarily parallel to its corresponding edge on the top face (due to different node displacements during the force-based smoothing procedure). Next, for elements involved in an edge flip during the interval $[t, t+\Delta t]$, each can be extruded together with the paired element it flipped an edge with, which locally forms a quadrangular prism with two reverse diagonals on the top and the bottom faces (figure 4.3, right). Again, similar to the unflipped case, the edges at Ω^t are not necessarily parallel to those at $\Omega^{t+\Delta t}$. However, for convenience in our notation, we will still refer to these vertically skew quadrilaterals as 'lateral faces' of the prisms. Finally, it is clear that the amount of node displacement during a time step must be limited to ensure sufficiently high element qualities. We control this dynamically by adjusting the size of the time step Δt and the pseudo time step δ in section 3.1 in order to avoid inverted prisms.

Similarly, we can generalize this prism formulation to our density control operations. Again for 2D, these operations do not change any mesh connectivities except for the group of elements involved in the operation. Locally, in the setting of the space-time framework, these elements can also be extruded together along the temporal dimension. When adding or removing a node within a single element, it forms a triangular prism with three interior edges on its top or bottom face; for edge splitting and edge collapsing on a pair of triangles, a quadrangular prism is again created with a few interior edges on both the top and the bottom faces. So in summary, the point-wise connection strategy described above produces a mesh of triangular and quadrangular prisms based on our moving-mesh algorithm.

Note that if there is no mesh connectivity or mesh density change, we can simply solve the problem using triangular prisms, which is consistent with the idea looming behind the method-of-lines. But for large deformations, we cannot only use the space-time mesh consisting of triangular and quadrangular prisms. For element involved in edge flipping, on the top and the bottom faces of each quadrangular prism, the solution is not represented by a Galerkin expansion (see Eq 4.6) in a quadrilateral, but different polynomial spaces within two triangular elements. This means that quadrangular prisms must be triangulated in order to match the diagonals on these faces (i.e. spatial mesh connectivity changes). The same issue happens to the density control operations as well, since there must be some interior edges splitting the bottom/top faces. Without inserting extra mesh nodes, next we consider how to split these prisms into a conforming mesh of tetrahedra, by first describing how to perform a valid local triangulation of each prism, and second how to globally ensure that two adjacent prisms respect the same diagonal on their shared lateral face.

4.3.2 Local Triangulation of Prisms

Again, we will first study local triangulations without density control operations. Our goal is to form prism triangulations entirely based on the nodes in the given spatial meshes. First of all, we locally index the nodes of each prism in a counterclockwise order. As shown in figure 4.4, for each prism V between Ω^t and $\Omega^{t+\Delta t}$, if V a triangular prism, we locally number the vertices on the bottom face as $\{p_1^{V,t}, p_2^{V,t}, p_3^{V,t}\}$ and the vertices on its top face as $\{p_1^{V,t+\Delta t}, p_2^{V,t+\Delta t}, p_3^{V,t+\Delta t}\}$. Similarly, vertices of a quadrangular prism V on the bottom and

top faces are locally numbered as $\{p_1^{V,t}, p_2^{V,t}, p_3^{V,t}, p_4^{V,t}\}$ and $\{p_1^{V,t+\Delta t}, p_2^{V,t+\Delta t}, p_3^{V,t+\Delta t}, p_4^{V,t+\Delta t}\}$, respectively. In addition, without loss of generality, we require that the original shared edge on Ω^t is the line segment $\overline{p_2^{V,t} p_4^{V,t}}$ and the new shared edge on $\Omega^{t+\Delta t}$ is the line segment $\overline{p_1^{V,t+\Delta t} p_3^{V,t+\Delta t}}$.

We will denote by F_i^V the lateral face with vertices at $p_i^{V,t}$, $p_i^{V,t+\Delta t}$, $p_j^{V,t}$ and $p_j^{V,t+\Delta t}$, where $j = (i \mod n) + 1$, n is the number of lateral faces of V, and $1 \leq i \leq n$. For each lateral face F_i^V, there are two possible face diagonals which we define using a sign function $S_i^V(p_i^{V,t}, p_i^{V,t+\Delta t}, p_j^{V,t}, p_j^{V,t+\Delta t})$ for each F_i^V according to

$$S_i^V(p_i^{V,t}, p_i^{V,t+\Delta t}, p_j^{V,t}, p_j^{V,t+\Delta t}) = \begin{cases} -1 & \text{if the diagonal edge is } \overline{p_i^{V,t} p_j^{V,t+\Delta t}} \\ +1 & \text{if the diagonal edge is } \overline{p_i^{V,t+\Delta t} p_j^{V,t}} \end{cases} \tag{4.29}$$

for $1 \leq i \leq n$.

Now, a triangulation of a triangular prism V is completely determined by the values of its 3 sign functions S_1^V, S_2^V and S_3^V. Combinatorially, it is easy to see that there are $2^3 = 8$ different combinations, but only 6 of these give valid triangulations (illustrated in figure 4.4, left). Note that the two uniform cases $\{S_1^V = +1, S_2^V = +1, S_3^V = +1\}$ and $\{S_1^V = -1, S_2^V = -1, S_3^V = -1\}$ cannot be used for valid triangulations.

For the quadrangular case, we first make the following definition:

Definition 4.1. *For a quadrangular prism V, we define the* **standard value** *of the sign function S_i^V as $+1$ if i is odd and -1 if i is even.*

Since a quadrangular prism V has 4 lateral faces, a triangulation is determined by the values of the 4 corresponding sign functions S_1^V, S_2^V, S_3^V and S_4^V, for a total of $2^4 = 16$ different combinations. However, in order allow for a valid triangulation of V, a combination of sign functions must satisfy the following condition:

Condition 1. *There are at least two consecutive sign functions S_i^V and $S_{mod(i,4)+1}^V$ which are set to their standard values.*

Geometrically, this condition means at least one of the 4 tetrahedra $\{p_1^{V,t}, p_2^{V,t}, p_4^{V,t}, p_1^{V,t+\Delta t}\}$, $\{p_2^{V,t}, p_3^{V,t}, p_4^{V,t}, p_3^{V,t+\Delta t}\}$, $\{p_2^{V,t}, p_1^{V,t+\Delta t}, p_2^{V,t+\Delta t}, p_3^{V,t+\Delta t}\}$ or $\{p_4^{V,t}, p_1^{V,t+\Delta t}, p_3^{V,t+\Delta t}, p_4^{V,t+\Delta t}\}$ must be formed and included in the final triangulation. This results in a total of 9 possible combinations of S_1^V, S_2^V, S_3^V and S_4^V that correspond to valid triangulations of V (illustrated in figure 4.4, right). The local triangulations are summarized in table 4.1.

Lastly, we consider the triangulation of the prisms corresponding to our density control operations. First, no matter whether we add or remove a node within a single triangle, we can locally extrude that triangle to form a triangular prism. Note that in the previous cases (corresponding to the element without edge flipping or density change), if the combination $\{S_1^V, S_2^V, S_3^V\}$ gives a valid triangulation, then both the top and the bottom surface must become a surface of a tetrahedron. Comparing with these cases, the density control operation

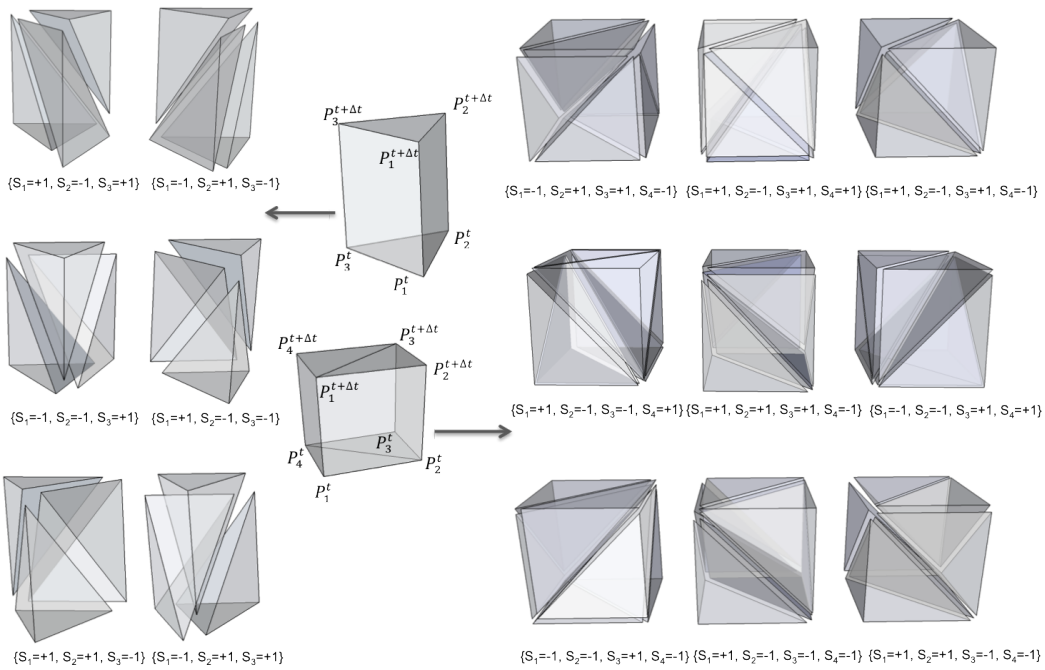


Figure 4.4: All the valid triangulations of a triangular prism (left) and of a quadrangular prism (right). The sets below each triangulation show the values of the corresponding sign functions.

Summary for Triangular Prism				
$\{S_1^V, S_2^V, S_3^V\}$	$\{+1,+1,+1\}$	$\{+1,+1,-1\}$	$\{+1,-1,+1\}$	$\{+1,-1,-1\}$
Valid?	No	Yes	Yes	Yes
$\{S_1^V, S_2^V, S_3^V\}$	$\{-1,+1,+1\}$	$\{-1,+1,-1\}$	$\{-1,-1,+1\}$	$\{-1,-1,-1\}$
Valid?	Yes	Yes	Yes	No
Summary for Quadrangular Prism				
$\{S_1^V, S_2^V, S_3^V, S_4^V\}$	$\{+1,+1,+1,+1\}$	$\{+1,+1,+1,-1\}$	$\{+1,+1,-1,+1\}$	$\{+1,+1,-1,-1\}$
Valid?	No	Yes	Yes	Yes
$\{S_1^V, S_2^V, S_3^V, S_4^V\}$	$\{+1,-1,+1,+1\}$	$\{+1,-1,+1,-1\}$	$\{+1,-1,-1,+1\}$	$\{+1,-1,-1,-1\}$
Valid?	Yes	Yes	Yes	Yes
$\{S_1^V, S_2^V, S_3^V, S_4^V\}$	$\{-1,+1,+1,+1\}$	$\{-1,+1,+1,-1\}$	$\{-1,+1,-1,+1\}$	$\{-1,+1,-1,-1\}$
Valid?	No	Yes	No	No
$\{S_1^V, S_2^V, S_3^V, S_4^V\}$	$\{-1,-1,+1,+1\}$	$\{-1,-1,+1,-1\}$	$\{-1,-1,-1,+1\}$	$\{-1,-1,-1,-1\}$
Valid?	No	Yes	No	No

Table 4.1: The summary of sign function value combination for local triangulations.

in a single element only splits the top/bottom surface of the prism. So we can simply triangulate the tetrahedron involving that top/bottom surface by connecting the added/removed node to the tetrahedron vertex not on the top/bottom surface. In the left of figure 4.5, we give an example of node removal in a single element.

On the other hand, the elements involved in the edge splitting or edge collapsing cases form a quadrangular prism similar to the edge flipping cases. The difference is that on the top/bottom surface, the former prisms have a diagonal but the current ones have 4 short edges from each surface vertex connected to a central node, where without loss of generality we can assume there are two diagonals crossing each other. Similar to the triangular case, if the combination $\{S_1^V, S_2^V, S_3^V, S_4^V\}$ gives a valid triangulation for a quadrangular prism corresponding to the edge flipping operation, the two triangles sharing the diagonal edge on the top/bottom surface must belong to 2 different tetrahedra. Now when we add another diagonal to either surface, we can turn these 2 tetrahedra into 4 small ones by connecting the central node to the tetrahedron vertices not on the top/bottom surface (as shown on the right of figure 4.5).

In conclusion, we can see that the summary in table 4.1 is also suitable to the prisms formed by the density control operations. Therefore, without loss of generality, we will discuss the algorithm in the next section assuming that we move the mesh only using node movement and edge flipping, in order to simplify the description.

Figure 4.5: Examples of triangulated prisms corresponding to local density control operations.

4.3.3 Diagonal Matching and the Global Algorithm

The last step is to obtain a global tetrahedral triangulation from the extruded prism elements between Ω^t to $\Omega^{t+\Delta t}$. This is nontrivial as the local triangulations are not independent, because each prism should match the diagonals of shared lateral faces with their neighboring prisms.

First of all, if we consider a space-time mesh only consisting of triangular prisms (i.e. only involving density control in single elements), it is not difficult to match the diagonals based on the analysis of section 4.3.2. An algorithm similar to the natural switch (the technique we used in section 4.2.2 to assign the value of switch function for the CDG method) can be directly applied to this problem (we will discuss the details in section 4.4.1). However, due to the more complicated triangulation rules for quadrangular prisms, it turns out to

be more challenging to work out an algorithm covering all the possible cases. Here we describe an efficient depth-first algorithm which finds a global triangulation that satisfies these restrictions.

Note that for each interior lateral face, there are two sign functions belonging to elements on each side. The algorithm will set the value of each sign function iteratively but set the value of the two sign functions corresponding to the same lateral face simultaneously in order to satisfy the condition of diagonal matching. Before describing the algorithm, we introduce the following definitions:

Definition 4.2. *For a prism V, let V^* be the adjacent prism of V with $F_{i^*}^{V^*} = F_i^V$ for an index i^*. We say F_i^V is a* wall *if the values of S_i^V and $S_{i^*}^{V^*}$ are both set and $S_i^V = S_{i^*}^{V^*}$. We say F_i^V is* accessible *if the values of S_i^V and $S_{i^*}^{V^*}$ are both unset.*

During the algorithm, we will make the assumption that each prism V only has three possible states as follows,

Triangular Prism at State 1. *$\{F_1^V, F_2^V, F_3^V\}$ are all accessible;*

Triangular Prism at State 2. *Exactly one of $\{F_1^V, F_2^V, F_3^V\}$ has become a wall and the other two are accessible;*

Triangular Prism at State 3. *$\{F_1^V, F_2^V, F_3^V\}$ have all become walls and $\{S_1^V, S_2^V, S_3^V\}$ can make a valid triangulation of V.*

Quadrangular Prism at State 1. *$\{F_1^V, F_2^V, F_3^V, F_4^V\}$ are all accessible;*

Quadrangular Prism at State 2. *Exactly one of the pair faces $\{F_1^V, F_3^V\}$ and $\{F_2^V, F_4^V\}$ have both become walls, at least one of two corresponding S_i^V was set to its standard value, and both F_i^V in the other pair are accessible.*

Quadrangular Prism at State 3. *$\{F_1^V, F_2^V, F_3^V, F_4^V\}$ have all become walls and $\{S_1^V, S_2^V, S_3^V, S_4^V\}$ can make a valid triangulation of V.*

With this assumption, we now introduce the algorithm by its three main operations.

Operation 1: Optimal Local Triangulation of Prisms
Based on the assumption, throughout the algorithm, if V has not been triangulated it must be at state 1 or 2. We then choose an optimal local triangulation of V by

$$\arg\max_{\mathcal{T}^V} \min_{K \in \mathcal{T}^V} Q(K) \tag{4.30}$$

where $\mathcal{T}^V$ denotes the set of all the possible valid triangulations of V, whose sign functions respect the ones prescribed on the walls. Again, $Q(K)$ represents the quality of each tetrahedron K of $\mathcal{T}^V$, which is calculated by the measure proposed in Eq 3.6.

From the local triangulations in figure 4.4, it can easily be verified that $\mathcal{T}^V$ is nonempty when V is at state 1 or 2. In other words, we can always find a valid triangulation of V.

Operation 2: Sign Function Synchronization of Neighbor Prisms
When a prism V is triangulated by operation 1, in order to not violate the assumption made at the beginning, we have to transfer V to state 3. Therefore, as operation 2, we start from each accessible F_i^V, and update the sign functions of the corresponding neighbor prisms to make F_i^V a wall. For instance, suppose F_i^V was accessible before the local triangulation of V, and V^* is the adjacent prism with $F_{i^*}^{V^*} = F_i^V$ for some index i^*. Again, according to the assumption, V^* must be in state 1 or 2 since $F_{i^*}^{V^*}$ was accessible before the triangulation of V. So in total, there are 4 possible cases for V^*:

Case 1. *If V^* is triangular at state 1, we simply set $S_{i^*}^{V^*} = S_i^V$, which makes F_i^V and $F_{i^*}^{V^*}$ walls and transfers V^* to state 2;*

Case 2. *If V^* is quadrangular at state 1, we set $S_{i^*}^{V'} = S_i^V$ and $S_{mod(i^*+1,4)+1}^{V^*}$ to their standard values. This transfers V^* into state 2. If V^{**} is the adjacent prism of V^* with shared face $F_{mod(i^*+1,4)+1}^{V^*}$, then we continue to update sign functions of V^{**} recursively using operation 2;*

Case 3. *Suppose V^* is triangular at state 2 with a wall $F_{j^*}^{V^*}$ for some $j^* \neq i^*$. We then use operation 1 to triangulate V^* immediately under the restrictions imposed by the prescribed values of $S_{j^*}^{V^*}$ and $S_{i^*}^{V^*} = S_i^V$. Let k^* be the third index other than i^* and j^* and V^{**} be the adjacent prism of V^* with shared face $F_{k^*}^{V^*}$. To transfer V^* to state 3, we have to continue updating sign functions of V^{**} recursively using operation 2;*

Case 4. *Suppose V^* is quadrangular at state 2 with a pair of opposite walls, say, $S_{j^*}^{V^*}$ and $S_{k^*}^{V^*}$ (where $i^* \neq j^*$ and $i^* \neq k^*$). Let l^* be the fourth index other than i^*, j^* and k^*. Again, we use operation 1 to triangulate V^*, under the restrictions imposed by the prescribed values of $S_{j^*}^{V^*}$ and $S_{k^*}^{V^*}$, as well as $S_{i^*}^{V^*} = S_i^V$. In spite of having three restrictions, we claim that the equation (4.30) is always solvable for V^*. To prove this, we first note that either $S_{j^*}^{V^*}$ or $S_{k^*}^{V^*}$ has been set to their standard values since V^* is at state 2. Therefore, as long as $S_{l^*}^{V'}$ is set to the standard value, the combination of sign functions must satisfy Condition 1 and thus gives a valid local triangulation. Finally, similarly to the previous case, if V^{**} is the adjacent prism of V^* with the shared face $F_{l^*}^{V^*}$, we continue to update sign functions of V^{**} recursively using operation 2, in order to turn $F_{l^*}^{V^*}$ into a wall and thereby transfer V^* to state 3.*

Operation 3: Triangulation Adjustment of Root Prism
As shown in figure 4.6, if we triangulate a prism V by operation 1 and repeatedly encounter the cases $2-4$ when synchronizing sign functions of neighbors by operation 2, then a path will be made which we will refer to as an 'updating path'. In fact, every updating path will eventually end with one of three possibilities: 1. a prism belonging to Case 1 (figure 4.6, right); 2. a domain boundary; 3. back to the root prism V from a face which is not yet a wall (figure 4.6, left). The third case is the only potentially difficult one, since the last prism

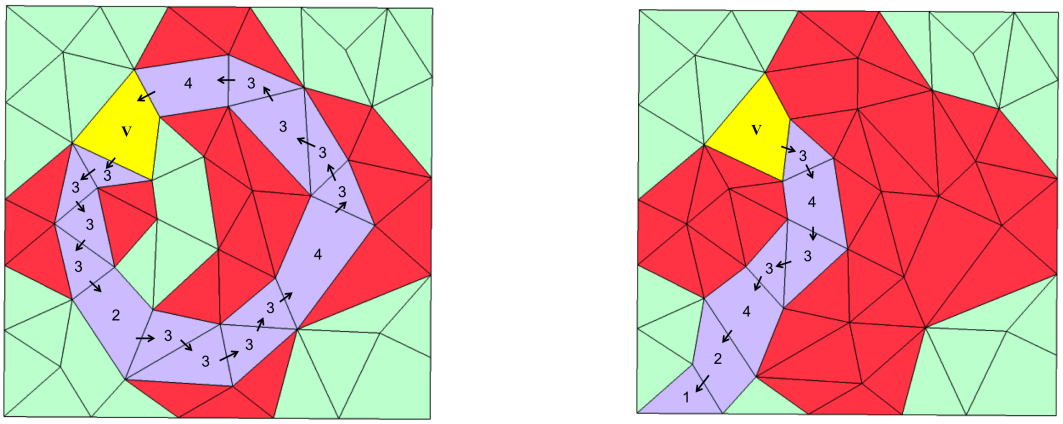


Figure 4.6: Two examples of updating paths. Each triangle represents a triangular prism and each quadrilateral represents a quadrangular prism. The yellow element is the root prism V, the green elements are prisms at state either 1 or 2, and the red elements are already triangulated, i.e., at state 3. The purple elements denote an updating path directed by the black arrows. The corresponding case number that each purple element belongs to is also shown. The example path on the left ends when it returns to V, and the example path on the right ends with a triangular prism belonging to Case 1.

of an updating path is a neighbor of the root prism V, but they may have inconsistent values of the sign functions corresponding to their shared face. Suppose the last prism is V' with the shared face $F_{i'}^{V'} = F_i^V$ but $S_{i'}^{V'} \neq S_i^V$. Operation 3 is to change the value of S_i^V to that of $S_{i'}^{V'}$ and thus make both $F_{i'}^{V'}$ and F_i^V into walls.

It is clear from the local triangulations derived in Section 4.3.2 that changing values of S_i^V might result in a new combination of sign functions the does not correspond to a valid local triangulation of the root prism V. We will avoid this situation by arranging the order by which new updating paths are launched.

First, we consider a triangular root prism V. If V was at state 1 before its triangulation, at least two updating paths will be launched from V. If the first two updating paths are launched from faces with different values of their sign functions, then there will always be a valid triangulation of V regardless of whether any path will return to V or the value of the third sign function will be changed. In fact, the condition above can always be satisfied if the first updating path is launched from the face with a sign function value different from the other two. Similarly, if V was at state 2 with a wall F_i^V before its triangulation, operation 3 will not destroy the validity of the local triangulation provided that the first updating path is launched from a face with sign function value different from S_i^V.

Next we consider the case that the root prism V is quadrangular. Recall that for a valid triangulation of V to exist, it is required that there are two consecutive sign functions set to their standard values. If V was at state 2 before its triangulation, it must have a wall, say F_i^V, whose sign function was set to the standard value. After triangulation by operation

Algorithm 4.1 Space-Time Mesh Generation with Path Marching

Require: A spatial mesh $MESH1$ of Ω^t and $MESH2$ of $\Omega^{t+\Delta t}$
Ensure: A space-time mesh $STMESH$ of $\Omega[t, t+\Delta t]$
  Create prisms by extruding elements from $MESH1$ to $MESH2$ and make a list of prisms called $PList$
  Initialize an empty list $STMESH$ for storing the elements of the space-time mesh
  while $PList$ is non-empty **do**
    Pop a prism V from $PList$
    if V is not at state 3 **then**
      Triangulate V by operation 1
      Make a list of F_i^V which has not been a wall, called $FList$
      Sort $FList$ by the order of launching updating paths discussed for operation 3
      for F_i^V in $FList$ **do**
        Find the neighbor prism $NBPrism$ adjacent to V by F_i^V ▷ Initialization of an updating path
        while $NBPrism$ exists (not exist if encountering domain boundary) and is not V **do**
          Synchronize sign functions of $NBPrism$ by operation 2
          if $NBPrism$ belongs to Case 2-4 **then**
            Update $NBPrism$ by operation 2 and continue the updating path
          else
            Break ▷ The updating path ends with a $NBPrism$ of Case 1
          end if
        end while
        if $NBPrism$ is V **then** ▷ The updating path back to the root prism
          Adjust the values of sign functions of V by operation 3 if necessary
        end if
      end for
    end if
    Push all the elements from the resultant triangulation of V into $STMESH$
  end while
  return $STMESH$

1, there will be another face adjacent to F_i^V, say F_j^V, whose sign function is also set to the standard value. When the first updating path from F_j^V is launched, Condition 1 will then not be violated even if operation 3 is applied. Finally, if V was in state 1, to respect Condition 1 we have to launch the first two updating paths from faces with standard sign function values. This can always be done, since no face of V is a wall and thus any one can be changed to the standard value if necessary.

Based on the three operations described above, the full algorithm is summarized in algorithm 4.1. The algorithm looks quite complicated since there are a number of different situations to be considered and addressed. However, in practice the entire algorithm only depends on combinatoric properties. Although there are a few conditional sentences to adjust the values of the sign functions, most operations are assignments and there are no arithmetic operations. Therefore, the algorithm is fairly efficient even with a large number of quadrangular prisms (i.e. many edge flippings, splittings and collapsings).

Recall that we described the operations based on the assumption that throughout the algorithm, each prism only has three possible states. In fact, initially all prisms faces are set to state 1; and as the algorithm progresses, the operations in the algorithm can only change

a prism into state 2 or 3. Finally, the global tetrahedral triangulation is complete if and only if all the prism faces have become walls. Therefore, by induction, it is clear that the assumption holds for all prisms and that the algorithm will return a global triangulation of the space-time domain $\Omega[t, t + \Delta t]$. For more details of this space-time mesh generation for 2D in space, we refer to [84, 85].

4.4 Space-Time Mesh Generation for 3D Problems

For 3D problems, the space-time mesh has to be created in 4 dimensions (3D in space plus 1D in time). Clearly the mesh generation becomes more challenging due to a number of reasons: first of all, the spatial moving mesh strategy involves more possible local operations; second, we lack a sufficient understanding of the geometric structures for simplex meshes in 4D; finally, visualization seems almost impossible for mesh quality validation. Nevertheless, from 3D to 4D, we notice that many combinatoric properties of simplices remain the same, which inspires us to think of an algorithm in 3D first by purely combinatoric analysis, and then generalize it to 4D. Here we present our algorithm for 4D space-time mesh generation and prove its validity.

4.4.1 An Alternative Algorithm for 2D Problems

Recall that we try to minimize the number of mesh nodes and elements when generating the space-time meshes, in order to reduce the computational cost of solving implicit systems. Algorithm 4.1 in section 4.3 is able to create meshes without inserting any other nodes other than those on the spatial meshes, however, the path-marching technique seems quite difficult to generalize to higher dimensions. For this reason, here we propose an alternative algorithm by relaxing our requirement and allowing a few nodes to be added between two consecutive spatial meshes.

At the beginning of section 4.3.3, we briefly described an idea that if there are no quadrangular prisms, we can simply index the mesh nodes and for each lateral face, choose the diagonal connecting the node with the smaller index on the bottom spatial mesh, to that with larger index on the top spatial mesh. More precisely, using the same notation as in section 4.3.3, we propose an indexing approach that globally indexes all the mesh nodes and for each lateral face, chooses the diagonal $\overline{p_i^{V,t} p_j^{V,t+\Delta t}}$ whenever the global index of local node i is smaller than that of node j. As for this approach, we can give the following theorem:

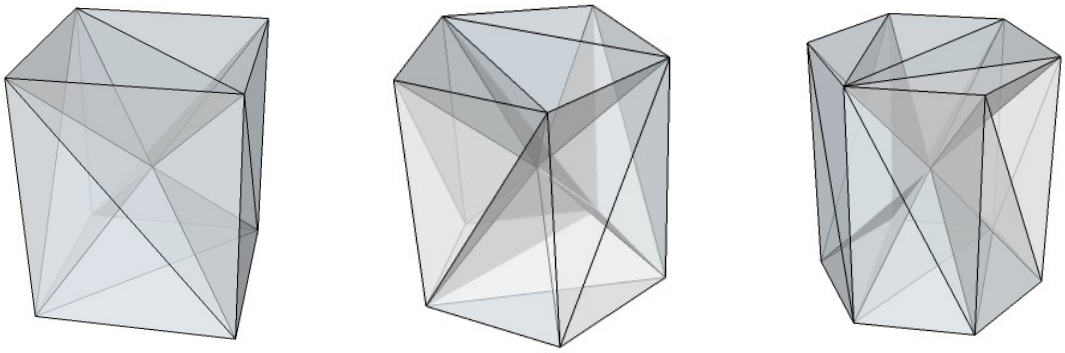
Theorem 4.1. *If the prism mesh is constructed without quadrangular prisms, the indexing approach can always triangulate the prism mesh into a valid tetrahedra mesh.*

Proof. First, we consider the lateral face shared by any of two prisms V and V'. Without loss of generality, we assume this face is F_1^V in V and $F_1^{V'}$ in V'. Then according to the way we numbered the local prism vertices, we must have $p_1^{V,t} = p_2^{V',t}$ and $p_1^{V',t} = p_2^{V,t}$. Therefore,

the diagonals of F_1^V and $F_1^{V'}$ are both based on the comparison between the global indices of $p_1^{V,t}$ and $p_2^{V,t}$, which proved that these two diagonals must match each other.

Second, we prove that those assigned diagonals result in valid triangulation for all the prisms. Assume that there exists one triangular prism V which fails to have a valid triangulation. Denote the global indices of $p_1 V, t$, $p_2^{V,t}$ and $p_3^{V,t}$ as I_1, I_2 and I_3, respectively. According to table 4.1, it means that we have either $I_1 > I_2 > I_3 > I_1$ or $I_1 < I_2 < I_3 < I_1$, which reaches a contradiction. □

Unfortunately, the problem with this indexing algorithm is that the diagonal combination may not give a valid triangulation for each local quadrangular prism. Motivated by this, we look for a triangulation solution which works for arbitrary diagonal combinations of polygonal prisms. Note that for any polygonal prism, if all its faces have been triangulated (these surface triangulations do not necessarily provide a valid triangulation of the prism), we can always insert an interior node within the prism (e.g. at the center of the prism) and achieve a valid prism triangulation by connecting it to all the prisms vertices. In figure 4.7, we show how to apply this idea to quadrangular, pentagonal and hexagonal prisms. Using this, the triangulation of quadrangular prisms corresponding to edge flipping turns out to be very straightforward. Also, it is not difficult to employ this idea for prisms corresponding to edge splitting or collapsing operations. For such prisms, again we insert a center node and connect it to all nodes on the surface (note that there might be some additional nodes on the surface besides the prism vertices).



Quadrangular Case Pentagonal Case Hexagonal Case

Figure 4.7: Triangulation of polygonal prisms with an additional interior node.

One important step of this approach is how to determine the position of the extra inserted node. Assume that the space-time coordinates of the inserted nodes is $(\boldsymbol{x}_e, t_e)$. Then in order to produce a valid local triangulation of any polygonal prism without inverted tetrahedra, the coordinates of the inserted node must have the following property:

For any τ such that $t \leq \tau \leq t + \Delta t$, denote the intersection polygon of the polygonal prism on time level τ as $\boldsymbol{P}_\tau$, the spatial part $\boldsymbol{x}_e$ of the coordinates must satisfy the following conditions:

- *$\boldsymbol{x}_e$ must be within $\boldsymbol{P}_\tau$.*
- *If $\boldsymbol{x}_e$ is connected to all the vertices of $\boldsymbol{P}_\tau$, the resulting triangulation must be valid without inverted triangles.*

In practice, the temporal part t_e is often chosen as $t + \frac{1}{2}\Delta t$. Furthermore, we find that since we can control the mesh motion and avoid large displacements by adjusting the timestep Δt and the pseudo-timestep δ, the intersection polygons $\boldsymbol{P}_\tau$ have very similar shape for all $t \leq \tau \leq t + \Delta t$. Therefore, as a approximation, we choose the spatial part of the coordinates $\boldsymbol{x}_e$ such that $\boldsymbol{x}_e$ only has to satisfy the two properties above for polygon $\boldsymbol{P}_{t+\frac{1}{2}\Delta t}$.

It is easy to see that if polygon $\boldsymbol{P}_{t+\frac{1}{2}\Delta t}$ is convex, any interior point of this polygon satisfies the conditions above. For example, the centroid of $\boldsymbol{P}_{t+\frac{1}{2}\Delta t}$ is one possible candidate. However, when $\boldsymbol{P}_{t+\frac{1}{2}\Delta t}$ is non-convex, the centroid may not satisfy the second requirement at all times (See Figure 4.8).

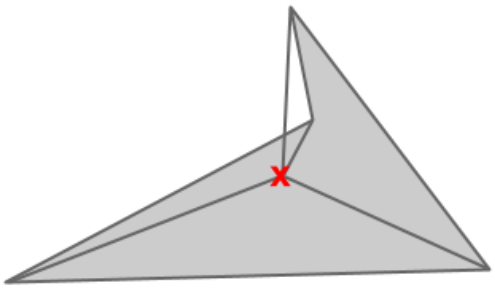


Figure 4.8: An example of an invalid triagulation for a non-convex polygon. The point marked in red is the inserted node.

Motivated by this, we first notice that each edge of the polygon determine an infinite line in the plane. In order to avoid inverted elements, $\boldsymbol{x}_e$ must be chosen such that $\boldsymbol{x}_e$ is always on the same side as the inside normal direction of this line. Mathematically, if the line is expressed by $\boldsymbol{a} \cdot \boldsymbol{x} = b$ for some constant $\boldsymbol{a}$ and b, the constraint above can be represented by a linear inequality as $\boldsymbol{a} \cdot \boldsymbol{x}_e \leq b$. If we apply this constraint to all the edges. Then we can express a system of linear inequalities by

$$\boldsymbol{A}\boldsymbol{x}_e \leq \boldsymbol{b}, \quad \boldsymbol{A} \in \mathbb{R}^{n_f \times d} \quad \boldsymbol{b} \in \mathbb{R}^{n_f} \tag{4.31}$$

where n_f is the number of edges. Note that in our problem, $\boldsymbol{A}$ and $\boldsymbol{b}$ can be determined by the geometry of the polygon $\boldsymbol{P}_{t+\frac{1}{2}\Delta t}$, and if $\boldsymbol{x}_e$ is chosen such that Eq 4.31 is satisfied, we can make sure that the local polygonal prism can be validly triangulated with the inserted node.

To further improve the mesh quality, assume that by connecting the vertices of $\boldsymbol{P}_{t+\frac{1}{2}\Delta t}$ to $\boldsymbol{x}_e$, we obtain a triangulation of $\boldsymbol{P}_{t+\frac{1}{2}\Delta t}$ as $\mathcal{T}^h = \{K\}$ and we determine $\boldsymbol{x}_e$ by solving the following optimization problem with linear constraints:

$$\begin{aligned} &\min_{\boldsymbol{x}_e} \sum_{K\in\mathcal{T}^h} \frac{1}{Q(K)^4} \\ &s.t.\ \boldsymbol{A}\boldsymbol{x}_e \leq \boldsymbol{b} \end{aligned} \tag{4.32}$$

where $Q(K)$ is the element quality defined by Eq 3.6.

Note that in 2D, edge flippings only happen within convex quadrilaterals, but when density control is implemented for pairs of triangles, edge splitting and collapsing may happen within non-convex quadrilaterals. More importantly, the use of Eq 4.32 is particularly critical for 4D space-time mesh generation since 3D local mesh topology changes may happen frequently within non-convex polyhedra. We will get back to this discussion later.

Next, we continue using the notations from section 4.3.2. By combining this center-node insertion method with the indexing idea, we reach a new algorithm of space-time mesh generation for 2D problems, shown in algorithm 4.2. In summary, the main idea for this algorithm has two parts: first, the diagonals assigned by our indexing approach guarantee valid triangulation of all the triangular prisms; second, an additional node is inserted within each quadrangular prism to help complete a valid triangulation.

Although the idea of algorithm 4.2 appears more clear and simple than algorithm 4.1, we have to mention that algorithm 4.2 may create many more tetrahedra than algorithm 4.1. For example, for a quadrangular prism corresponding to edge flipping operation, algorithm 4.1 is able to locally triangulate it using 5 or 6 tetrahedra; on the other hand, algorithm 4.2 has to use 12 tetrahedra to triangulate the same prism. This means that for complicated geometric motions that need a number of local connectivity changes, the space-time mesh from algorithm 4.2 may have about twice as many elements as that from algorithm 4.1. Moreover, since both algorithms mainly depend on combinatoric properties, the computational efficiency is roughly comparable. Therefore, in practice, algorithm 4.1 is chosen for our numerical tests in two spatial dimensions.

However, algorithm 4.2 turns to be much more attractive when considering 4D space-time mesh generation, which we will discuss next.

4.4.2 Generalization of Prisms, Lateral Faces and Diagonals

Before moving to our algorithm for 4D space-time meshes, we need some more understanding about geometries in 4 dimensions, analogous to our procedures in section 4.3. First of all, we can understand the geometric structure of a triangular prism in a way of point-wise extrusion, that it is formed by extruding a 2D simplex from t to $t + \Delta t$. To generalize this idea, we can extrude a d-dimensional simplex from t to $t + \Delta t$ to form a $(d + 1)$-dimensional prism. For example, when $d = 1$, we can extrude a line segment with unit length to form

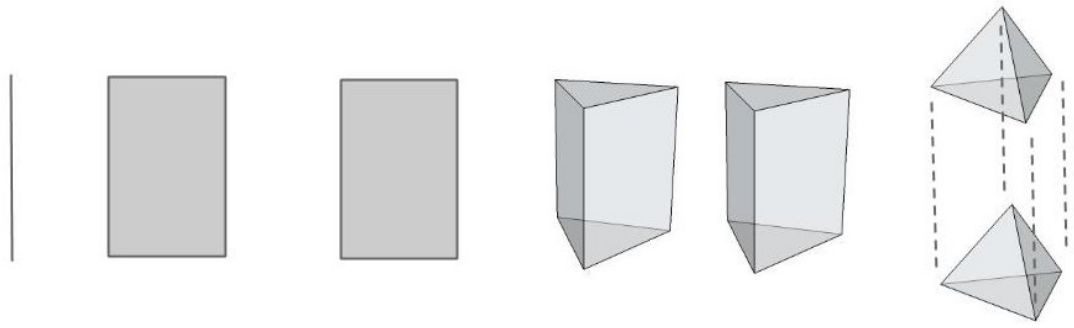
Algorithm 4.2 Space-Time Mesh Generation with Insertion of Additional Nodes

Require: A spatial mesh $MESH1$ of Ω^t and $MESH2$ of $\Omega^{t+\Delta t}$
Ensure: A space-time mesh $STMESH$ of $\Omega[t, t+\Delta t]$
Create prisms by extruding elements from $MESH1$ to $MESH2$ and make a list of prisms called $PList$.
Index all the nodes on Ω^t except those to be removed from $\Omega^{t+\Delta t}$ by our local density control operations.
while $PList$ is non-empty **do**
Pop a prism V from $PList$
if V is triangular **then**
Set $n = 3$.
else
Set $n = 4$.
Set an additional node $p^{V,t+\frac{1}{2}\Delta t}$ whose coordinates are determined by solving Eq 4.32.
Connect $p^{V,t+\frac{1}{2}\Delta t}$ to all the nodes on the surface of V.
end if
for i from 1 to n **do**
Set $j = (i \mod 3) + 1$
Set the global indices of $p_i^{V,t}$ and $p_j^{V,t}$ as I_1 and I_2, respectively.
if $I_1 < I_2$ **then**
Set diagonal of face F_i^V as $\overline{p_i^{V,t} p_j^{V,t+\Delta t}}$.
else
Set diagonal of face F_i^V as $\overline{p_j^{V,t} p_i^{V,t+\Delta t}}$.
end if
end for
Push all the elements from the resultant triangulation of V into $STMESH$
end while
return $STMESH$

a 2D 'prism' (i.e. a rectangle). When $d = 3$, we extrude a tetrahedra to form a so-called 4D-prism.

Moreover, we can further generalize the definition of the 'lateral face' of a $(d+1)$-dimensional prism. In figure 4.9, we can see that a rectangle has lateral faces as two line segments, which can be treated as 1D prisms; and for a triangular prism, it has three lateral faces as 2D prisms. Using the same logic, any d-dimensional prism should have d lateral faces as $(d-1)$-dimensional prisms.

Finally, we also have to generalize the definition of diagonals of lateral faces. As the bottom and top 'faces' of a $(d+1)$-dimensional prism V are two d-dimensional simplices, we denote the vertices of these simplices as $\{p_1^{V,t}, p_2^{V,t}, \dots p_d^{V,t}\}$ and $\{p_1^{V,t+\Delta t}, p_2^{V,t+\Delta t}, \dots p_d^{V,t+\Delta t}\}$,



2D 'Prism' 3D Prism 4D 'Prism'

Figure 4.9: 'Prisms' and 'Lateral Faces' in 2D, 3D and 4D. In each case, the right is a 'prism' and the left is one of its 'lateral faces'.

	2D Prism	3D Prism	4D Prism
Geometry of Bottom/Top Face	Line Segment	Triangle	Tetrahedra
Geometry of Lateral Faces	Line Segment	Rectangle	Triangular prism
Number of Lateral Faces	2	3	4
Number of Diagonals	$\binom{2}{2}$	$\binom{3}{2}$	$\binom{4}{2}$

Table 4.2: Summary of the geometric properties for prisms in 2D, 3D and 4D.

then the diagonal can be understood as any line segment with the following form

$$\overline{p_i^{V,t} p_j^{V,t+\Delta t}} \quad i \neq j \tag{4.33}$$

To complete the diagonal assignment in $d+1$ dimensions, we mean that for each pair $\{i, j\}$ when $1 \leq i < j \leq d$, we choose one of the two possible diagonals $\overline{p_i^{V,t} p_j^{V,t+\Delta t}}$ and $\overline{p_i^{V,t+\Delta t} p_j^{V,t}}$. In this way, we know that we need assign $\binom{d}{2}$ diagonals for a $(d+1)$-dimensional prism for triangulation, which is consistent with the number of edges of a d-dimensional simplex. Lastly, we summarize these geometric properties in table 4.2.

4.4.3 Extension to 3D Problems

In this section, we prove that algorithm 4.2 can be directly applied to the case with three spatial dimensions plus one temporal dimension.

First, recall that when considering tetrahedral space-time meshes, we extrude elements to form either a triangular prism or a quadrangular prism using the idea of point-wise extrusion. For 3D spatial meshes, the mesh nodes still have the same point-wise relationship (except those nodes added or removed by local density change operations). Therefore, if we connect every mesh node from the tetrahedral spatial mesh of the 3D domain Ω^t to that of $\Omega^{t+\Delta t}$, geometrically, we again form a space-time mesh using 4D-prisms. Here, for simplicity of the description, we introduce the definitions of two different 4D-prisms:

Definition 4.3. *We define* simple 4D-prisms *as those formed by extruding a single tetrahedron which is not involved in any local operations, and define* complex 4D-prisms *as those formed by extruding a group of tetrahedra involved in the same local connectivity or density change operation.*

Note that simple 4D-prisms are simply the regular prisms that we discussed in section 4.4.2. Next, we discuss how to apply algorithm 4.2 to 4D space-time mesh generation with two steps.

Step 1: Indexing and the Triangulation of Simple 4D-Prisms
First, we use exactly the same indexing approach to assign diagonals in the 4D case as that in section 4.4.1. Consider any simple or complex 4D prism, with vertices denoted by $\{p_1^{V,t}, p_2^{V,t}, \dots p_m^{V,t}\}$ and $\{p_1 V, t+\Delta t, p_2^{V,t+\Delta t}, \dots p_m^{V,t+\Delta t}\}$ ($m > 4$ for complex 4D-prisms). We then globally index all the mesh nodes (expect those added or removed by density change operations), and denote the global indices of $\{p_1^{V,t}, p_2^{V,t}, \dots p_m^{V,t}\}$ as $\{I_1, I_2, \dots I_m\}$. We connect $\overline{p_i^{V,t} p_j^{V,t+\Delta t}}$ if $I_i < I_j$, or $\overline{p_j^{V,t} p_i^{V,t+\Delta t}}$ if $I_i > I_j$. Then we can prove the following theorem:

Theorem 4.2. *If the 4D prism mesh is constructed purely by simple 4D-prisms, the indexing approach can always triangulate the prism mesh into a valid tetrahedral mesh. More precisely, in each simple 4D-prism, if we have the ordered vertices $\{p_{(1)}^{V,t}, p_{(2)}^{V,t}, p_{(3)}^{V,t}, p_{(4)}^{V,t}\}$ with $I_{(1)} < I_{(2)} < I_{(3)} < I_{(4)}$, the simple 4D-prism is triangulated by the following four 4D-simplices with vertex sets*

$$\begin{aligned} T_1 &= \{p_{(1)}^{V,t}, p_{(1)}^{V,t+\Delta t}, p_{(2)}^{V,t+\Delta t}, p_{(3)}^{V,t+\Delta t}, p_{(4)}^{V,t+\Delta t}\}, \quad T_2 = \{p_{(1)}^{V,t}, p_{(2)}^{V,t}, p_{(2)}^{V,t+\Delta t}, p_{(3)}^{V,t+\Delta t}, p_{(4)}^{V,t+\Delta t}\} \\ T_3 &= \{p_{(1)}^{V,t}, p_{(2)}^{V,t}, p_{(3)}^{V,t}, p_{(3)}^{V,t+\Delta t}, p_{(4)}^{V,t+\Delta t}\}, \quad T_4 = \{p_{(1)}^{V,t}, p_{(2)}^{V,t}, p_{(3)}^{V,t}, p_{(4)}^{V,t}, p_{(4)}^{V,t+\Delta t}\} \end{aligned} \tag{4.34}$$

Proof. First, by theorem 4.1, we can see: 1. For the lateral face (i.e. the triangular prism) shared by any of two simple prisms V and V', the three assigned diagonals on the side of V are respectively matched by those assigned from V'. 2. The assigned three diagonals must provide a valid triangulation on that lateral face. Moreover, as a preliminary check, it is easy to verify that for each vertex set in (4.34), any two vertices are connected based on our indexing approach, indicating that the four 4D-simplices in (4.34) have really been formed.

Second, we show that these 4D-simplices form a valid triangulation of any simple prism. We consider a reference simple prism with

$$\begin{aligned} &p_{(1)}^{V,t} = \{0,0,0,0\} \quad , p_{(1)}^{V,t+\Delta t} = \{0,0,0,1\} \quad , p_{(2)}^{V,t} = \{1,0,0,0\} \quad , p_{(2)}^{V,t+\Delta t} = \{1,0,0,1\} \\ &p_{(3)}^{V,t} = \{0,1,0,0\} \quad , p_{(3)}^{V,t+\Delta t} = \{0,1,0,1\} \quad , p_{(4)}^{V,t} = \{0,0,1,0\} \quad , p_{(4)}^{V,t+\Delta t} = \{0,0,1,1\} \end{aligned} \tag{4.35}$$

For any d-dimensional simplex $T = \{p_1, p_2, \dots, p_{d+1}\}$, the volume $\mathrm{Vol}(T)$ is given by

$$\mathrm{Vol}(T) = |\frac{1}{d!} \det\left(p_2 - p_1, p_3 - p_1, \dots, p_{d+1} - p_1\right)| \tag{4.36}$$

Denote the bottom face of this reference 4D-prism as T^* with vertex set $\{p^{V,t}_{(1)}, p^{V,t}_{(2)}, p^{V,t}_{(3)}, p^{V,t}_{(4)}\}$. Since the 'height' of the reference 4D-prism is 1, we know that the volume of the reference 4D-prism is equal to $\mathrm{Vol}(T^*)$. Based on Eq 4.36, a simple computation shows that

$$\mathrm{Vol}(T^*) = \sum_{i=1}^{4} \mathrm{Vol}(T_i). \tag{4.37}$$

Therefore, we know that the volume of four 4D-simplices is equal to the volume of the reference prism. Next, we show that the four 4D-simplices do not have intersection, that is,

$$\mathrm{Vol}(T_i \cap T_j) = 0, \quad \forall 0 < i < j \leq 4 \tag{4.38}$$

Based on the coordinates in Eqs 4.35, we express any convex combination p_i of vertices of 4D-simplex T_i as

$$p_1 = a_1 p^{V,t}_{(1)} + b_1 p^{V,t+\Delta t}_{(1)} + c_1 p^{V,t+\Delta t}_{(2)} + d_1 p^{V,t+\Delta t}_{(3)} + e_1 p^{V,t+\Delta t}_{(4)} = (c_1, d_1, e_1, b_1 + c_1 + d_1 + e_1) \tag{4.39}$$

$$p_2 = a_2 p^{V,t}_{(1)} + b_2 p^{V,t}_{(2)} + c_2 p^{V,t+\Delta t}_{(2)} + d_2 p^{V,t+\Delta t}_{(3)} + e_2 p^{V,t+\Delta t}_{(4)} = (b_2 + c_2, d_2, e_2, c_2 + d_2 + e_2) \tag{4.40}$$

$$p_3 = a_3 p^{V,t}_{(1)} + b_3 p^{V,t}_{(2)} + c_3 p^{V,t}_{(3)} + d_3 p^{V,t+\Delta t}_{(3)} + e_3 p^{V,t+\Delta t}_{(4)} = (b_3, c_3 + d_3, e_3, d_3 + e_3) \tag{4.41}$$

$$p_4 = a_4 p^{V,t}_{(1)} + b_4 p^{V,t}_{(2)} + c_4 p^{V,t}_{(3)} + d_4 p^{V,t}_{(4)}, + e_4 p^{V,t+\Delta t}_{(4)} = (b_4, c_4, d_4 + e_4, e_4) \tag{4.42}$$

If we require that all the coefficients are positive (i.e. exclude the case when the point is on the surface), then we can summarize the possibilities as follows:

- If any point can be expressed by Eq 4.39, it cannot be expressed by Eq 4.40, 4.41 or 4.42, since the sum of the first three components of the coordinate is smaller than the last component in Eq 4.39 but it is greater in the other Eqs.
- If any point can be expressed by Eq 4.40, it cannot be expressed by Eq 4.41 or 4.42, since the sum of the second and the third components of the coordinate is smaller than the last component in Eq 4.40 but it is greater in the other two Eqs.
- Any point cannot be expressed by both Eq 4.41 and 4.42, since the third components of the coordinate is smaller than the last component in Eq 4.41 but it is greater in Eq 4.42.

Therefore, we conclude that Eq 4.38 holds for the vertices in Eqs 4.35. Combining Eq 4.37 and 4.38, we have proved that T_1, T_2, T_3 and T_4 construct a valid triangulation of the reference simple 4D-prism.

For an arbitrary simple 4D-prism (including those mildly distorted by the DistMesh mesh-node movement), suppose there exists a smooth mapping to transform it into the reference 4D-prism. Since there are no mesh topology (connectivity) changes during this mapping, this finally shows that T_1, T_2, T_3 and T_4 construct a valid triangulation of an arbitrary simple 4D-prism with mild distorion.

□

Step 2: Node Insertion and the Triangulation of Complex 4D-Prisms
According to theorem 4.2, we know that the indexing approach from algorithm 4.2 can be directly applied to the 4D cases and it creates valid a triangulation for simple 4D-prisms. However, complex 4D-prisms may not be able to be triangulated by the assigned diagonals. Therefore we again study their triangulation using additional node insertion.

As illustrated in figure 4.7, we split our approach of triangulating complex 4D-prisms into two stages: 1. triangulate all the surfaces; 2. insert one node and connect it to all the surface nodes. First, we have to understand the triangulation of the surfaces in the first stage. Since these complex 4D-prisms are associated with either local connectivity changes or density control operations, from figure 3.3 and 3.5, we can see that the top and bottom 'faces' of a complex 4D-prisms are always triangulated polyhedra. As for the lateral faces, first we can see that according to the summary of the simple 4D-prisms in table 4.2, the number of lateral faces for a complex 4D-prism should be equal to the number of boundary faces of the top/bottom polyhedron and all the lateral faces are simply triangular prisms. For example, recall that our connectivity or density change operations in 3D have similar structures with a k-sided polygon in the middle and two nodes on each side of it. For such structures, the 4D-prism has $2k$ lateral faces in total.

As the second stage, we can use the similar approach to determine the position of the inserted node as that in 2D. First, it is not difficult to see that for any τ such that $t \leq \tau \leq t+\Delta t$, the intersection geometry of a complex 4D-prism at time level τ is a polyhedron. Then if the spatial part of the coordinates of the inserted node is again $\boldsymbol{x}_e$, for any polyhedron intersection $\boldsymbol{P}_\tau$, it must satisfy

- *$\boldsymbol{x}_e$ must be within $\boldsymbol{P}_\tau$.*
- *If $\boldsymbol{x}_e$ is connected to all the vertices of $\boldsymbol{P}_\tau$, the resulting triangulation is valid without inverted tetrahedra.*

Again, as a approximation, we only have to find $\boldsymbol{x}_e$ satisfying these two conditions for polyhedron intersection $\boldsymbol{P}_{t+\frac{1}{2}\Delta t}$. Similar to the 2D case, the centroid of $\boldsymbol{P}_{t+\frac{1}{2}\Delta t}$ can be chosen for convex polyhedra but it might not even be inside $\boldsymbol{P}_{t+\frac{1}{2}\Delta t}$ for a non-convex case. Therefore, a more robust algorithm to find $\boldsymbol{x}_e$ is to solve the optimization problem Eq 4.32. The only difference is that in 3D, each linear inequality constraint is associated with a boundary face of $\boldsymbol{P}_{t+\frac{1}{2}\Delta t}$, where for each plane determined by a boundary face of $\boldsymbol{P}_{t+\frac{1}{2}\Delta t}$, $\boldsymbol{x}_e$ must be on the same side as the inside face normal direction.

Finally, using exactly the same procedure as the proof of theorem 4.2, we can find a reference complex 4D-prism corresponding to each local operation and prove: 1. our indexing idea can ensure the valid triangulation for each lateral face; 2. The volume sum of 4D-simplices created by this node insertion idea is equal to the volume of the reference 4D-prism; 3. any two of these resultant 4D-simplices have zero-volume intersection. With these properties, we can conclude that our indexing idea plus node-insertion technique is able to deliver a unstructured 4D space-time mesh with 4D-simplices.

Chapter 5

Arbitrary Lagrangian-Eulerian Discontinuous Galerkin Methods with Local L^2 Projections

In chapter 4, we described our space-time framework with a straightforward discretization of the unified spatial and temporal dimensions. As an alternative, here we discuss another strategy for addressing the large deformations using a mapping-based Arbitrary Lagrangian-Eulerian (ALE) framework.

5.1 Arbitrary Lagrangian-Eulerian Formulation

As illustrated in figure 5.1, we denote the time-varying domain as $\boldsymbol{v}(t) \in \mathbb{R}^n$. Our ALE formulation chooses a reference domain as $\boldsymbol{V} = \boldsymbol{V}(\boldsymbol{X})$, which is fixed at all times, and constructs a smoothly differentiable mapping $\boldsymbol{x}(\boldsymbol{X}, t) : \boldsymbol{V} \to \boldsymbol{v}(t)$ from the reference domain to the moving domain, such that every point $\boldsymbol{X} \in \boldsymbol{V}$ is mapped to a point $\boldsymbol{x}(\boldsymbol{X}, t) \in \boldsymbol{v}(t)$. Using the procedures from [57], we can define the deformation gradient $\boldsymbol{G}$ and mapping Jacobian g as

$$\boldsymbol{G} = \nabla \boldsymbol{x}, \quad g = \det \boldsymbol{G} \tag{5.1}$$

In the geometric sense, $\boldsymbol{G}$ measures how much the domain $\boldsymbol{v}(t)$ deforms and g measures the volume change due to the deformation. Besides, another important quantity for the domain motion is the mapping velocity $\boldsymbol{\nu}$, defined as follows

$$\boldsymbol{\nu} = \frac{\partial \boldsymbol{x}}{\partial t}. \tag{5.2}$$

Finally, since the boundary integrals and integration by substitution are frequently used in the derivations below, it is important to understand the elemental transformation in advance. As shown in figure 5.1, we denote the normal vectors on $\boldsymbol{v}(t)$ and $\boldsymbol{V}$ by $\boldsymbol{n}$ and $\boldsymbol{N}$, respectively.

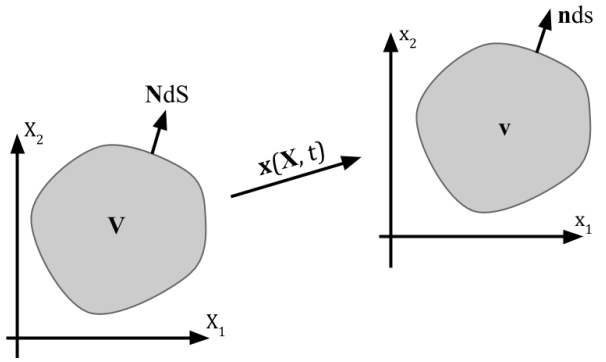


Figure 5.1: The mapping between the reference domain and the physical domain in the ALE framework.

Then the relationship between the elemental area and the element volume can be derived by Nanson's formula:

$$(d\boldsymbol{x}\cdot\boldsymbol{n})ds = dv = gdV = g(d\boldsymbol{X}\cdot\boldsymbol{N})dS = g((\boldsymbol{G}^{-1}d\boldsymbol{x})\cdot\boldsymbol{N})dS = (d\boldsymbol{x}\cdot(g\boldsymbol{G}^{-T}\boldsymbol{N}))dS \quad (5.3)$$

where since $d\boldsymbol{x} \neq 0$, we have $\boldsymbol{n}ds = g\boldsymbol{G}^{-T}\boldsymbol{N}dS$.

We can use these quantities and relationships to rewrite the conservation law from the physical domain $\boldsymbol{v}(t)$ into a new system in the reference domain $\boldsymbol{V}$. We again consider the compressible Navier-Stokes equations in $\boldsymbol{v}(t)$ as a system of conservation laws as in Eq 2.7,

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla_{\boldsymbol{x}}\cdot\boldsymbol{F}_{\boldsymbol{x}}^{\text{inv}}(\boldsymbol{u}) = \nabla_{\boldsymbol{x}}\cdot\boldsymbol{F}_{\boldsymbol{x}}^{\text{vis}}(\boldsymbol{u}, \nabla_{\boldsymbol{x}}\boldsymbol{u}). \quad (5.4)$$

Note that here the gradient $\nabla_{\boldsymbol{x}}$ is corresponding to the variables $\boldsymbol{x}$ for $\boldsymbol{v}(t)$. We then consider the integral form of Eq 5.4 and by the Reynolds transport theorem [11], we obtain

$$\begin{aligned}
\int_{\boldsymbol{v}(t)} \frac{\partial \boldsymbol{u}}{\partial t} dv &= \frac{\partial}{\partial t}\int_{\boldsymbol{v}(t)} \boldsymbol{u}dv - \int_{\partial \boldsymbol{v}} \boldsymbol{u}(\boldsymbol{\nu}\cdot\boldsymbol{n})ds \\
&= \int_{\boldsymbol{V}} \frac{\partial (g\boldsymbol{u})}{\partial t} dV - \int_{\partial \boldsymbol{V}} \boldsymbol{u}(\boldsymbol{\nu}\cdot(g\boldsymbol{G}^{-T}\boldsymbol{N}))dS \\
&= \int_{\boldsymbol{V}} \frac{\partial (g\boldsymbol{u})}{\partial t} dV - \int_{\partial \boldsymbol{V}} (g(\boldsymbol{u}\otimes\boldsymbol{\nu})\boldsymbol{G}^{-T})\cdot\boldsymbol{N})dS \\
&= \int_{\partial \boldsymbol{V}} (\boldsymbol{F}_{\boldsymbol{x}}^{\text{vis}} - \boldsymbol{F}_{\boldsymbol{x}}^{\text{inv}})\cdot(g\boldsymbol{G}^{-T}\boldsymbol{N}))dS \\
&= \int_{\partial \boldsymbol{V}} (g(\boldsymbol{F}_{\boldsymbol{x}}^{\text{vis}} - \boldsymbol{F}_{\boldsymbol{x}}^{\text{inv}})\boldsymbol{G}^{-T})\cdot\boldsymbol{N})dS \quad (5.5)
\end{aligned}$$

Therefore, by introducing a new gradient $\nabla_{\boldsymbol{X}}$ corresponding to the variables $\boldsymbol{X}$ for $\boldsymbol{V}$, we get the modified version of a system of conservation laws in $\boldsymbol{V}$ as

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla_{\boldsymbol{X}} \cdot \boldsymbol{F}_{\boldsymbol{X}}^{\text{inv}}(\boldsymbol{U}) = \nabla_{\boldsymbol{X}} \cdot \boldsymbol{F}_{\boldsymbol{X}}^{\text{vis}}(\boldsymbol{U}, \nabla_{\boldsymbol{X}} \boldsymbol{U}), \tag{5.6}$$

where the new solution $\boldsymbol{U}$ and the corresponding inviscid and viscous fluxes can be written as,

$$\boldsymbol{U} = g\boldsymbol{u}, \;\; \boldsymbol{F}_{\boldsymbol{X}}^{\text{inv}} = g(\boldsymbol{F}_{\boldsymbol{x}}^{\text{inv}} - \boldsymbol{u} \otimes \boldsymbol{\nu})\boldsymbol{G}^{-T}, \;\; \boldsymbol{F}_{\boldsymbol{X}}^{\text{vis}} = g\boldsymbol{F}_{\boldsymbol{x}}^{\text{vis}}\boldsymbol{G}^{-T} \tag{5.7}$$

and by the chain rule, we also have,

$$\nabla_{\boldsymbol{x}} \boldsymbol{u} = (\nabla_{\boldsymbol{X}}(g^{-1}\boldsymbol{U}))\boldsymbol{G}^{-1} = (g^{-1}\nabla_{\boldsymbol{X}}\boldsymbol{U} + \boldsymbol{U} \otimes \nabla_{\boldsymbol{X}}(g^{-1}))\boldsymbol{G}^{-1}. \tag{5.8}$$

We refer to Ref [57] for more details on the derivation of this transformation.

5.2 Numerical Scheme

5.2.1 Discretization

We next describe the standard Discontinuous Galerkin (DG) formulation to solve the compressible Navier-Stokes equations in the reference domain $\boldsymbol{V}$. Again, we repeat the technique used in chapter 4. A standard procedure is used for the viscous terms, where the system 5.6 is split into a first-order system of equations:

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla_{\boldsymbol{X}} \cdot \boldsymbol{F}_{\boldsymbol{X}}^{\text{inv}}(\boldsymbol{U}) = \nabla_{\boldsymbol{X}} \cdot \boldsymbol{F}_{\boldsymbol{X}}^{\text{vis}}(\boldsymbol{U}, \boldsymbol{q}) \tag{5.9}$$

$$\nabla_{\boldsymbol{X}} \boldsymbol{U} = \boldsymbol{q}. \tag{5.10}$$

Similar to section 4.2, we introduce a triangulation $\mathcal{T}^h = \{K\}$ of the spatial reference domain $\boldsymbol{V}$ into elements K. On $\mathcal{T}^h$, we define the broken space $\mathcal{V}_T^h$ and Σ_T^h as [31],

$$\mathcal{V}_T^h = \{\boldsymbol{v} \in [L^2(\boldsymbol{V})]^d \mid \; \boldsymbol{v}|_K \in [\mathcal{P}_p(K)]^d \;\; \forall K \in \mathcal{T}^h\}, \tag{5.11}$$

$$\Sigma_T^h = \{\boldsymbol{\sigma} \in [L^2(\boldsymbol{V})]^{m \times d} \mid \; \boldsymbol{\sigma}|_K \in [\mathcal{P}_p(K)]^{m \times d} \;\; \forall K \in \mathcal{T}^h\} \tag{5.12}$$

where d is the spatial dimension, m is the number of components in solution $\boldsymbol{U}$, and $\mathcal{P}_p(K)$ denotes the space of polynomials of degree at most $p \geq 1$ on K. Then the DG formulation for equations (5.9) and (5.10) becomes: find $\boldsymbol{U}^h \in \mathcal{V}_T^h$ and $\boldsymbol{q}^h \in \Sigma_T^h$ such that for each $K \in \mathcal{T}^h$, we have

$$\begin{aligned} \int_K \frac{\partial \boldsymbol{U}^h}{\partial t} \cdot \boldsymbol{v}^h dx - \int_K \boldsymbol{F}_{\boldsymbol{X}}^{\text{inv}}(\boldsymbol{U}^h) : \nabla_{\boldsymbol{X}} \boldsymbol{v}^h \, dx + \oint_{\partial K} (\widehat{\boldsymbol{F}_{\boldsymbol{X}}^{\text{inv}} \cdot \boldsymbol{n}}) \cdot \boldsymbol{v}^h \, ds \\ = - \int_K \boldsymbol{F}_{\boldsymbol{X}}^{\text{vis}}(\boldsymbol{U}^h, \boldsymbol{q}^h) : \nabla_{\boldsymbol{X}} \boldsymbol{v}^h \, dx + \oint_{\partial K} (\widehat{\boldsymbol{F}_{\boldsymbol{X}}^{\text{vis}} \cdot \boldsymbol{n}}) \cdot \boldsymbol{v}^h \, ds, \quad \forall \boldsymbol{v}^h \in \mathcal{V}_T^h \end{aligned} \tag{5.13}$$

$$\int_K \boldsymbol{q}^h : \boldsymbol{\sigma}^h \, dx = - \int_K \boldsymbol{U}^h \cdot (\nabla_{\boldsymbol{X}} \cdot \boldsymbol{\sigma}^h) \, dx + \oint_{\partial K} (\widehat{\boldsymbol{U}^h} \otimes \boldsymbol{n}) : \boldsymbol{\sigma}^h \, ds, \qquad \forall \boldsymbol{\sigma}^h \in \Sigma_T^h. \tag{5.14}$$

For numerical fluxes, we again use Roe's method [68] to approximate the inviscid flux $\widehat{\boldsymbol{F}_{\boldsymbol{X}}^{\text{inv}} \cdot \boldsymbol{n}}$ (see Eq 4.10) and treat the viscous fluxes $\widehat{\boldsymbol{F}_{\boldsymbol{X}}^{\text{vis}} \cdot \boldsymbol{n}}$ and $\widehat{\boldsymbol{U}^h}$ using the Compact Discontinuous Galerkin (CDG) method proposed in [54] (see Eqs 4.24 and 4.25). Our formulation of Eqs 5.13 and 5.14 is based on method-of-lines. Recall that the CDG method combines Eqs 5.13 and 5.14 to remove the dependency of $\boldsymbol{q}^h$, which leads to a final non-linear semi-discrete system formulated only for $\boldsymbol{U}^h$.

5.2.2 Geometric Conservation Law

In addition to Eq 5.6, a special treatment is required since the formulation above is unable to obtain a sufficiently accurate approximation solution when the true solution is constant [75]. To address this so-called geometric conservation law (GCL) problem, we follow the technique from [57] and solve an additional equation on the reference domain at each time step by introducing a new approximation $\bar{g}$ to the Jacobian g as

$$\frac{\partial \bar{g}}{\partial t} - \nabla_{\boldsymbol{X}} \cdot (g \boldsymbol{G}^{-1} \boldsymbol{\nu}) = 0 \tag{5.15}$$

Note that the exact solution of Eq 5.15 is $\bar{g} = g$, but it can only solve for an approximation to g due to the numerical discretization. Then we solve a modified version of Eq 5.6

$$\frac{\partial \tilde{\boldsymbol{U}}}{\partial t} + \nabla_{\boldsymbol{X}} \cdot \tilde{\boldsymbol{F}}_{\boldsymbol{X}}^{\text{inv}}(\tilde{\boldsymbol{U}}) = \nabla_{\boldsymbol{X}} \cdot \tilde{\boldsymbol{F}}_{\boldsymbol{X}^{\text{vis}}}(\tilde{\boldsymbol{U}}, \nabla_{\boldsymbol{X}} \tilde{\boldsymbol{U}}), \tag{5.16}$$

where $\tilde{\boldsymbol{U}} = \bar{g}\boldsymbol{u}$, and the fluxes $\tilde{\boldsymbol{F}}_{\boldsymbol{X}}^{\text{inv}}$ and $\tilde{\boldsymbol{F}}_{\boldsymbol{X}}^{\text{vis}}$ are obtained by replacing $\boldsymbol{u}$ from Eqs 5.7 with $\tilde{\boldsymbol{U}}/\bar{g}$. Again, Eqs 5.16 are solved using the same descretization from section 5.2.1.

Lastly, note that the flux term in Eq 5.15 is independent of $\tilde{\boldsymbol{U}}$ and $\bar{g}$. Since most of our numerical tests are based on prescribed domain motions, g, $\boldsymbol{G}$ and ν can be calculated directly at each time step as well as the flux term. In practice, we consider the integral form of Eq 5.15 and discretize using the same DG approach. This leads to a set of ODEs with respect to $\bar{g}$.

5.2.3 Temporal Integration

Based on Eq 5.15 and 5.16, we have two nonlinear discretized systems at each time step,

$$\boldsymbol{M}_1 \dot{\bar{g}}^h = \boldsymbol{R}_1(\boldsymbol{X}) \tag{5.17}$$

$$\boldsymbol{M}_2 \dot{\boldsymbol{U}}^h = \boldsymbol{R}_2(\boldsymbol{U}^h) \tag{5.18}$$

where $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are mass matrices. There is a number of ODE solvers which can be used to solve the equations above and the Runge-Kutta Methods are popular choices. Eq 5.17 is trivial to solve and at each time step, we first solve $\bar{g}$ and then plug it into Eq 5.18. Due to stability issues, an implicit solver is preferred for Eq 5.18 to avoid severe constraints

0.43586652	0.43586652	0	0
0.71793326	0.28206674	0.43586652	0
1	1.20849665	-0.64436317	0.43586652
	1.20849665	-0.64436317	0.43586652

Table 5.1: Butcher's Array for the DIRK3 scheme.

on the timestep Δt. Among implicit Runge-Kutta Methods, we employ a parallel high-order diagonally implicit Runge-Kutta (DIRK) solver with the corresponding Butcher's array shown in table 5.1.

More precisely, each stage of the DIRK scheme can be expressed as

$$\boldsymbol{M}_2\boldsymbol{k}_i^{n+1} = \boldsymbol{R}_2(\boldsymbol{U}^n + \Delta t \sum_{j=1}^{i} a_{ij}\boldsymbol{k}_j^{n+1}) \tag{5.19}$$

Once again we implement the parallel Newton-GMRES solver to compute $\boldsymbol{k}_i^{n+1}$ iteratively as we mentioned in section 4.2.3. During hte $(s+1)$th iteration, the adjustment $\delta\boldsymbol{k}_i^{n+1}$ is defined by solving the linear system

$$(\boldsymbol{M}_2 - \frac{\partial \boldsymbol{R}_2}{\partial \boldsymbol{k}_i^{n+1}})\delta\boldsymbol{k}_i^{n+1} = \boldsymbol{R}_2(\boldsymbol{U}^n + \Delta t \sum_{j=1}^{i} a_{ij}\boldsymbol{k}_j^{n+1,(s)}) \tag{5.20}$$

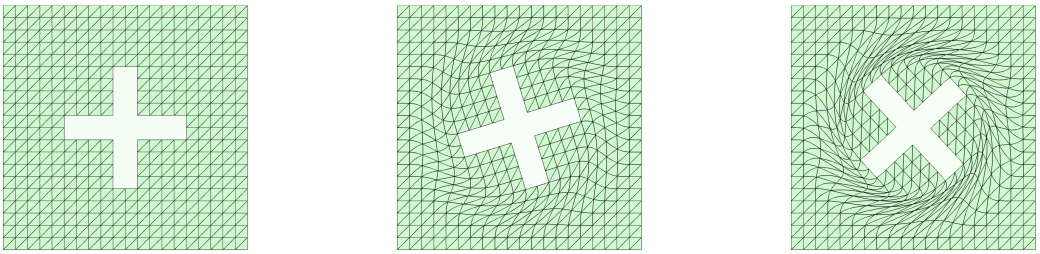
More details on the solver can be found in [58, 56].

5.3 Local L^2 Projections

Note that in the discrete ALE formulation from section 5.2, if we set the mesh velocity $\boldsymbol{\nu}$ to the velocity components in solution $\boldsymbol{U}^h$, the formulation becomes purely Lagrangian. [20] proposed a Lagrangian approach for fluid simulation with curvilinear finite elements, but a main difficulty for applying the fully Lagrangian framework to fluid problem is that the mesh can be easily entangled and destroyed due to complicated fluid motions such as vortex shedding.

Instead, the ALE framework gives more flexibility for choosing a smoother mapping $\boldsymbol{x}(\boldsymbol{X}, t)$, and it can construct the mesh motion independently of the solution vector $\boldsymbol{U}^h$. For this reason, the ALE framework appears quite attractive for flow problems on moving meshes. In many ALE simulations, the mesh is moved without connectivity changes for as many steps as possible and the transformed Navier-Stokes equations are solved in the reference domain based on well-conditioned smoothly differentiable mappings $\boldsymbol{x}(\boldsymbol{X}, t)$. But for problems involving large domain deformation, the mesh will eventually become poorly shaped due to the appearance of nearly inverted elements. As shown in figure 5.2, a cross is

placed in the center with a fixed outside square wall. The mesh quality decreases significantly as the cross starts spinning and the motion will finally invert the elements around the cross. To address this, remeshing can be used to replace the old triangulation $\mathcal{T}^h = \{K\}$ by a new one $\tilde{\mathcal{T}}^h = \{\tilde{K}\}$ in the reference domain $\boldsymbol{V}$ such that the mapping image of the new mesh in the physical domain $\boldsymbol{v}(t)$ maintains high quality. In addition to remeshing, we also need an efficient and accurate interpolation approach to transfer the solution between the old and the new meshes. Here, we focus on L^2 projections and introduce its mechanism next.



$t = 0$ $t = 0.4$ $t = 0.8$

Figure 5.2: Moving Meshes for a Spinning Cross without Topology Change. Three sample plots are given to show the mesh motion under the ALE framework.

5.3.1 Formulation

When we change mesh topology, the reference domain is still $\boldsymbol{V}$ but we created an alternative triangulation $\tilde{\mathcal{T}}^h$ on $\boldsymbol{V}$. Consider the broken spaces $\mathcal{V}^h_{\tilde{T}}$ and $\Sigma^h_{\tilde{T}}$, which are similar to the spaces defined in Eqs 5.11 and 5.12 but associated with $\tilde{\mathcal{T}}^h$. Recall that in section 4.2, we mentioned that the standard DG procedure expresses the numerical solution $\boldsymbol{U}^h$ as a linear combination of shape functions $\{\phi_1, \phi_2, \ldots, \phi_{N_p}\}$ on the broken space $\mathcal{V}^h_T$,

$$\boldsymbol{U}^h = \sum_{i=1}^{N_p} \boldsymbol{U_i}\phi_i. \tag{5.21}$$

In order to continue the time-stepping after remeshing, a new approximate solution to $\boldsymbol{U}^h \in \mathcal{V}^h_T$ is required corresponding to $\tilde{\mathcal{T}}^h$. Denote the shape functions of $\mathcal{V}^h_{\tilde{T}}$ by $\{\tilde{\phi}_1, \tilde{\phi}_2, \ldots, \tilde{\phi}_{\tilde{N}_p}\}$, and the L^2 projection of $\boldsymbol{U}^h$ onto $\mathcal{V}^h_{\tilde{T}}$ by $\tilde{\boldsymbol{U}}^h$. This projection satisfies the following property: for each $\tilde{K} \in \tilde{\mathcal{T}}^h$,

$$\int_{\tilde{K}} (\boldsymbol{U}^h - \tilde{\boldsymbol{U}}^h)\tilde{\phi}_i dx = 0, \qquad \forall i = 1, \ldots, \tilde{N}_p. \tag{5.22}$$

More precisely, if we also expand $\tilde{\boldsymbol{U}}^h$ by Galerkin formulation as

$$\tilde{\boldsymbol{U}}^h = \sum_{i=1}^{\tilde{N}_p} \tilde{\boldsymbol{U}}_{\boldsymbol{i}} \tilde{\phi}_i, \tag{5.23}$$

then Eq 5.22 can be written as

$$\begin{aligned} \sum_{i=1}^{\tilde{N}_p} \int_{\tilde{K}} \tilde{\boldsymbol{U}}_{\boldsymbol{i}} \tilde{\phi}_i \tilde{\phi}_j dx &= \sum_{i=1}^{N_p} \int_{\tilde{K}} \boldsymbol{U}_{\boldsymbol{i}} \phi_i \tilde{\phi}_j dx \\ &= \sum_{i=1}^{N_p} \sum_{K \in \mathcal{T}^h} \int_{\tilde{K} \cap K} \boldsymbol{U}_{\boldsymbol{i}} \phi_i \tilde{\phi}_j dx, \qquad \forall j = 1, \ldots, \tilde{N}_p. \end{aligned} \tag{5.24}$$

The Eq 5.24 result in a linear system,

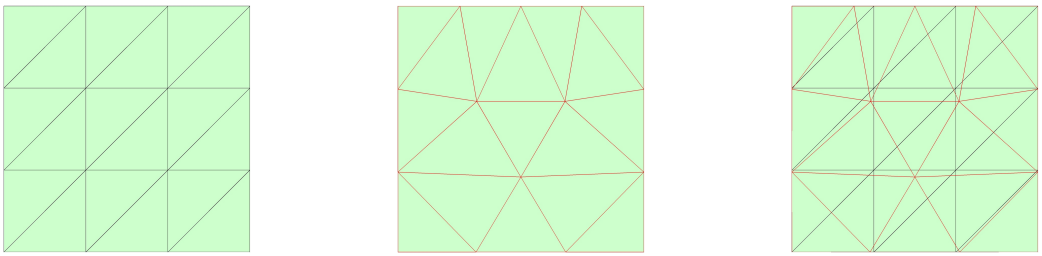
$$\boldsymbol{M} \tilde{\boldsymbol{U}}^h = \boldsymbol{P} \boldsymbol{U}^h \tag{5.25}$$

where we call $\boldsymbol{M}$ as mass matrix and $\boldsymbol{P}$ as projection matrix. They have the following form,

$$M_{j,i} = \int_{\tilde{K}} \tilde{\phi}_i \tilde{\phi}_j dx, \qquad P_{j,i} = \sum_{K \in \mathcal{T}^h} \int_{\tilde{K} \cap K} \phi_i \tilde{\phi}_j dx. \tag{5.26}$$

Eq 5.25 can be solved for $\tilde{\boldsymbol{U}}^h$ by simply inverting $\boldsymbol{M}$, and used as the transferred solution to resume the time-stepping process on the new triangulation $\tilde{\mathcal{T}}^h$. Note that the projection may lose accuracy if we compute the residual using the new solution $\tilde{\boldsymbol{U}}^h$, but later on our numerical results will show that the introduced error appears negligible and will not affect the overall order of convergence even with frequent projections. Moreover, one important advantage of using DG scheme is that unlike continuous Galerkin methods, DG defines both ϕ_i and $\tilde{\phi}_i$ as discontinuous element-wise polynomials, which implies that the mass matrix $\boldsymbol{M}$ has a block-wise format. This property allows us to locally compute each block of $\boldsymbol{M}$ within each $\tilde{K}$ and invert it at a low computational cost.

However, also because of this element-wise property, the second equal sign of Eq 5.24 indicates that each new element $\tilde{K}$ must be split into a collection of disjoint intersections $\tilde{K} \cap K$ for some $K \in \mathcal{T}^h$. Dealing with these projections directly for two arbitrary meshes poses several difficulties. For example, in figure 5.3, we create two independent meshes for the same unit square domain and try to do L^2 projection from mesh 1 to mesh 2 following Eq 5.24. First, we can see that an efficient and robust cut-cell algorithm is required to split the elements of mesh 2 in order to locate all the intersections $\tilde{K} \cap K$. In particular for 3D tetrahedral meshes, this is a fairly involved procedure. Second, again from figure 5.3, we notice that the intersections of simplex elements can have many possible shapes, and a sophisticated quadrature technique for arbitrary polygons or polyhedra must be incorporated for the evaluation of the volume integrals. Therefore, it can be concluded that L^2 projections for global remeshing is not an ideal approach for solution transfer.



Mesh 1 Mesh 2 Mixed Mesh

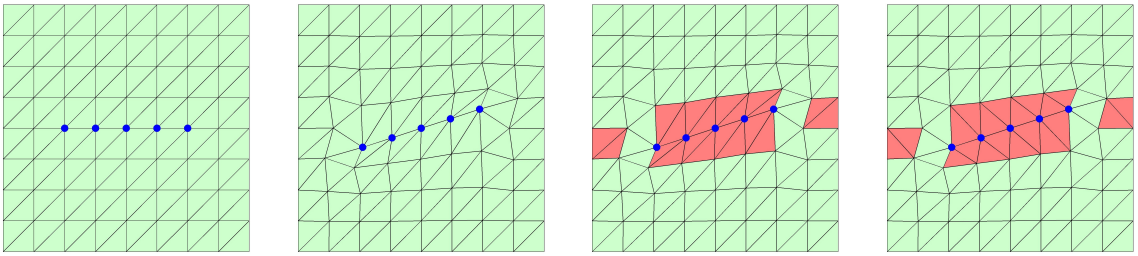
Figure 5.3: An Example of Global Remeshing. The 3rd plot overlaps mesh 1 with mesh 2.

Motivated by this, we try to combine our moving-mesh strategy from chapter 3 with the ALE framework. Our strategy can address the large deformations using local topology changes. Also, by taking advantage of our mesh moving techniques with local mesh operators and discontinuous element-wise polynomial solutions of the DG method, we can develop a simpler and more efficient algorithm to handle arbitrary deformations using local L^2 projections.

5.3.2 Implementation

The details of our method are described in algorithm 5.1. For simplicity, here we consider the algorithm without local density control operations. Then our mesh improvement strategy has two stages. The first stage only moves the nodes without any mesh topology changes. Therefore, it allows us to construct a smooth map and we can solve for one time step using a standard ALE method. Next, if needed, the second stage is to change the connectivity locally. In this stage, all the node positions are fixed before and after the connectivity change, so we can conduct our local L^2 projections within each group of flipped elements and transfer the solution efficiently. Unlike our space-time method, we use a while-loop for the connectivity change part in algorithm 5.1, indicating that multiple sweeps can be run under the condition that each element can be only involved in one operation of connectivity change. The algorithm repeats the two stages above and evolves the solution throughout the entire time period.

In 2D cases, it is clear that each local flipping operator replaces two old elements by two new elements, as shown in figure 5.5. In other words, the old pair $\{K_1, K_2\}$ and the new pair $\{\tilde{K}_1, \tilde{K}_2\}$ share the same group of vertices, so 4 sub-triangles $\{\tilde{K}_1 \cap K_1, \tilde{K}_1 \cap K_2, \tilde{K}_2 \cap K_1, \tilde{K}_2 \cap K_2\}$ are always formed as their intersections. The integrals in Eq 5.24 are then straightforward to evaluate based on these four sub-triangles and thus the components of solution $\boldsymbol{U}^h$ within two old elements are transferred into those of the new solution $\tilde{\boldsymbol{U}}^h$ with respect to the two new elements. Except for the pairs of flipped elements, all other components of $\boldsymbol{U}^h$ and $\tilde{\boldsymbol{U}}^h$ are unchanged. A 2D example is shown in figure 5.4 to demonstrate how one step of our algorithm performs.



Phase I Phase II Phase III Phase IV

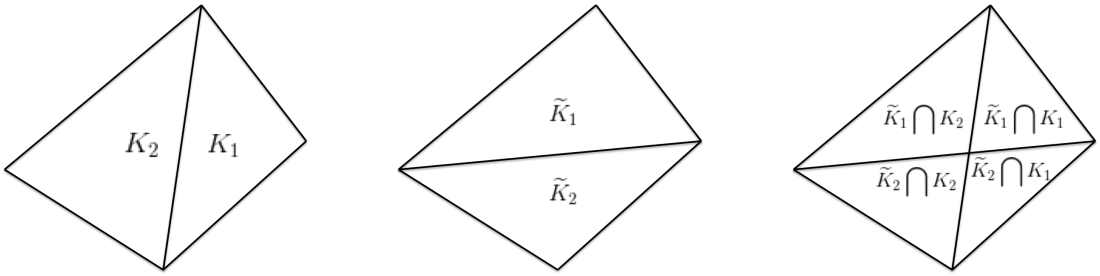
Figure 5.4: An illustration of the process in algorithm 5.1. Phase I: Select blue nodes to move; Phase II: Update interior nodes by force-based smoothing; Phase III: Select pairs of elements in red for flipping; Phase IV: Change the local connectivity within each pair of red elements.

Algorithm 5.1 Discontinuous Galerkin ALE Method with Local L^2 Projections

Require: Triangulation $\mathcal{T}^h$ and initial solution $\boldsymbol{U}^{h,t_0}$ at t_0
Require: Time step Δt and mesh quality threshold δ
Ensure: Solution $\boldsymbol{U}^{h,t_i}$ for each time step t_i until time T
  while $t_0 < t_i \leq T$ **do**
    Move the mesh by the DistMesh algorithm [60]
    Compute deformation gradient $\boldsymbol{G}$, mapping velocity $\boldsymbol{\nu}$ and mapping Jacobian g
    Solve $\boldsymbol{U}^{h,t_i}$ by the DG method with ALE framework
    while min quality of $K \in \mathcal{T}^h < \delta$ **do**
      Create $\tilde{\mathcal{T}}^h$ by local element flipping
      Solve for $\tilde{\boldsymbol{U}}^{h,t_i}$ by local L^2 projections
      $\mathcal{T}^h \leftarrow \tilde{\mathcal{T}}^h$
      $\boldsymbol{U}^{h,t_i} \leftarrow \tilde{\boldsymbol{U}}^{h,t_i}$
    end while
  end while

In 3D, similarly to the 2D cases, we also need to split the old and the new tetrahedra into sub-tetrahedra at the second stage and then implement the local L^2 projections by evaluating the integrals of Eq 5.24 in each of the sub-tetrahedra. This tetrahedral splitting is significantly more complicated than in 2D. However, since the connectivity changes are limited to a small group of tetrahedra, it is straightforward to analyze the resulting geometric structures and to consider all possible cases of splitting for each operation.

From figure 3.3, we can see that for any 3D operation, the old and the new groups of elements provide two different triangulations of a polyhedron with a special structure, which has a polygon (triangle, quadrilateral or pentagon) in the middle with one node on each side (top and bottom in the figure). By connecting these top and bottom nodes, we can find an intersection between the middle polygon and the connecting line (marked with a red cross



The old pair The new pair Sub-triangles

Figure 5.5: Local element splitting in 2D.

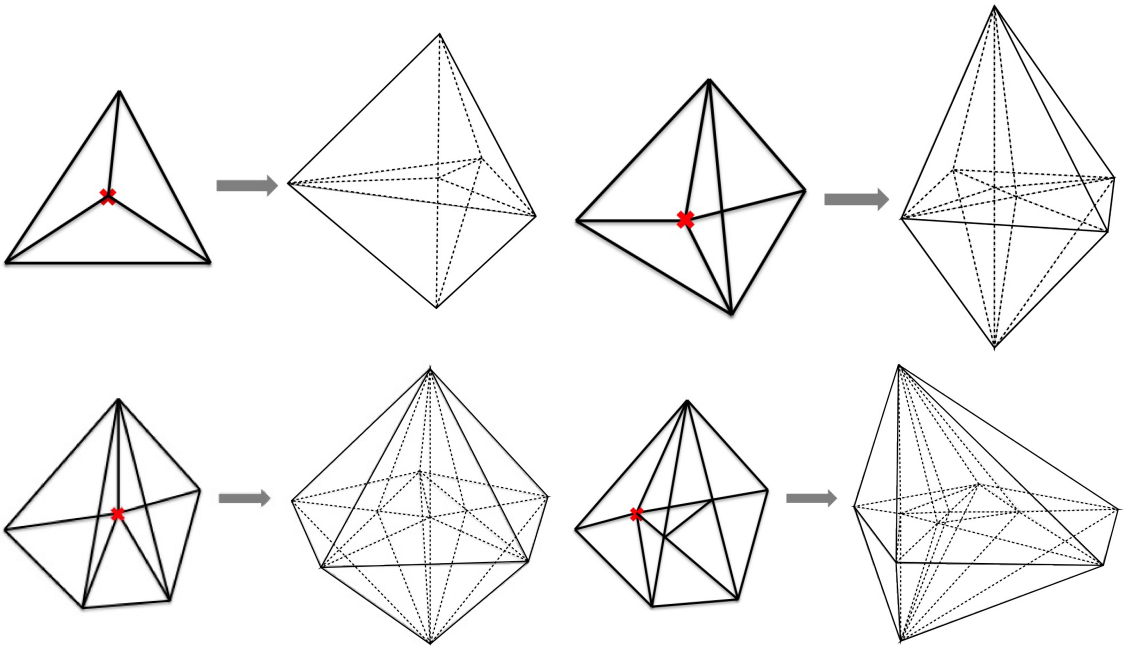


Figure 5.6: Tetrahedral splitting for 3D operations. In each plot, the triangulation of the middle polygon is on the left and the corresponding 3D triangulation is on the right. The red cross is the intersection between the middle polygon and the connecting line between the top and the bottom nodes.

in each plot of figure 5.6). Recall that we employ 4 different operations for 3D problems (see figure 3.3), and we can consider the splitting of each group of tetrahedra into sub-tetrahedra case by case. The resulting triangulation gives all the sub-tetrahedra needed for computing the integrals in Eq 5.24:

- Operation I and II are about the transformations between 2 and 3 tetrahedra. As

upper left plot in figure 5.6, we connect the center intersection to each vertex of the middle triangle, and connect all the middle nodes to the top and the bottom nodes.

- Operation III corresponds to transforming 4 elements into 4 new ones. The upper right plot in figure 5.6 shows that we can first triangulate the middle quadrilateral by assigning a diagonal. The intersection of the connecting line is then located in one of the two resultant triangles. Connect this intersection point to each vertex of the middle quadrilateral, and finally connect all the middle nodes to the top and to the bottom nodes.

- Operation IV turns 5 elements into 6 new ones. We first triangulate the middle pentagon by assigning two diagonals. The intersection of the connecting line is then in the middle triangle or in one of the two side triangles (Two different cases are shown in figure 5.6). Connect this intersection point to each vertex of the middle pentagon, and for the latter case, add another edge in addition to the diagonals and the edges connecting the intersection and the vertices, in order to complete a valid triangulation of the pentagon (see the last plot in figure 5.6). Finally connect all the middle nodes to the top and the bottom nodes.

Lastly, note that we have not included our density control operations in this ALE approach. But based on the analysis from chapter 3, it is not difficult to see that due to the same locality property, we can develop similar splitting strategies as in figure 5.5 and 5.6 to implement the local projections for these operations. In conclusion, the main idea behind our algorithm is to carry out local L^2 projections and avoid the implementation of the complicated and less accurate cut-cell algorithm. For more details about this local L^2 projection, we refer to [86].

Chapter 6

Numerical Results

In this chapter, we will validate our space-time and ALE methods via a number of numerical test cases. Through these tests, we try to show that our proposed methods not only provide solutions with high-order accuracy, but also have the ability to handle complex geometric motion. Furthermore, we also point out the potential use of our numerical schemes in engineering projects, by a practical application involving the simulation of multiple vertical axis wind turbines.

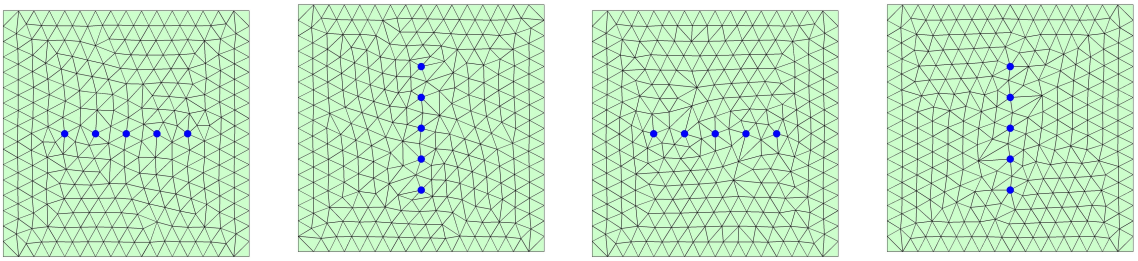
Note that all the simulations are carried out using the 3DG software[59], programmed in a mixed language model with Matlab, Python and C++ code.

6.1 Euler Vortex

As our first numerical experiment, we demonstrate the high-order accuracy of the numerical schemes by solving for a propagating compressible Euler vortex. Both the space-time and the ALE schemes are tested in 2D, and an additional 3D case is particularly designed for the ALE framework with local 3D L^2 projections.

6.1.1 2D Case

In 2D, we solve the Euler equations for a model problem of a compressible vortex in a 5-by-5 square domain and make a convergence test for both of our discontinuous Galerkin methods. The vortex is initially centered at $(x_0, y_0) = (0.8, 0.8)$ and moves to $(x_T, y_T) = (4.2, 4.2)$ with the free-stream at an angle $\theta = \pi/4$ with respect to the x-axis. At each (x, y, t), the



$\theta = 0$ $\theta = \frac{\pi}{2}$ $\theta = \pi$ $\theta = \frac{3\pi}{2}$

Figure 6.1: An illustration of the moving meshes. The five blues nodes are rotated about the center of the domain in a rigid manner, which induces other vertex movements and local element flipping. Note that θ is the degrees that the blue nodes have rotated about the center.

analytical solutions of velocity u and v, density ρ and pressure p are given by

$$\begin{aligned} u &= u_\infty(\cos\theta - \frac{\epsilon((y-y_0)-\bar{v}t)}{2\pi r_c}\exp(\frac{f(x,y,t)}{2})) \\ v &= u_\infty(\sin\theta + \frac{\epsilon((x-x_0)-\bar{u}t)}{2\pi r_c}\exp(\frac{f(x,y,t)}{2})) \\ \rho &= \rho_\infty(1 - \frac{\epsilon^2(\gamma-1)M_\infty^2}{8\pi^2}\exp(f(x,y,t)))^{\frac{1}{\gamma-1}} \\ p &= p_\infty(1 - \frac{\epsilon^2(\gamma-1)M_\infty^2}{8\pi^2}\exp(f(x,y,t)))^{\frac{\gamma}{\gamma-1}} \end{aligned} \tag{6.1}$$

where $f(x,y,t) = (1 - ((x-x_0)-\bar{u}t)^2 - ((y-y_0)-\bar{v}t)^2)/r_c^2$. Using the same notations as in chapter 2, we use $M_\infty = 0.1$ as the Mach number and $\gamma = 1.4$ as the adiabatic gas constant. $u_\infty = 1$, $p_\infty = (\gamma M_\infty^2)^{-1}$ and $\rho_\infty = 1$ are free-stream velocity, pressure and density, respectively. Moreover, $\bar{u}$ and $\bar{v}$ are the Cartesian components of the free-stream velocity with $\bar{u} = u_\infty\cos\theta$ and $\bar{v} = u_\infty\sin\theta$. The parameter $\epsilon = 2.0$ is the strength of the vortex and $r_c = 0.5$ is its size.

As a starting point, an unstructured mesh of the domain is created with element size h by DistMesh [60]. In order to show that our method remains high-order accurate even for large mesh deformations, we rotate some of the vertices counter-clockwise about the center of the domain with the angular velocity $\omega = 0.33$ rad/s. Since the walls of the domain are not moving, the rotation of the vertices induces large mesh deformations. To avoid inverted and low-quality elements, we improve the elements by our mesh node movement and edge flipping techniques. The process of this mesh motion is illustrated in figure 6.1.

Next we solve the Euler equations on this moving mesh until time $T = 3.4\sqrt{2}$ with a Dirichlet boundary condition given by the analytic solution. Due to our choice of ω, the rotating nodes rotate about 90 degrees within the time period T, which provides large enough deformations and a sufficient number of local mesh topology changes. We compare

the numerical results with the analytical solutions above. Some sample solutions of the pressure field are shown in figure 6.2.

Space-Time Method

For our space-time DG method, since it depends on a conforming unstructured space-time mesh without using the method of lines, we are mainly interested in its space-time convergence rather than merely the spatial one. To test this convergence, we first create two refinement sequences: one where we refine the mesh size h and fix $\Delta t << h$ and another where we refine Δt and fix $h << \Delta t$. We then compare the numerical errors based on the h and Δt from these sequences and eventually find that the errors are almost equal when h from the first sequence is roughly the same as Δt from the second sequence, which numerically shows that the spatial and temporal parts of the errors are comparable when $h \approx u_\infty \Delta t = \Delta t$.

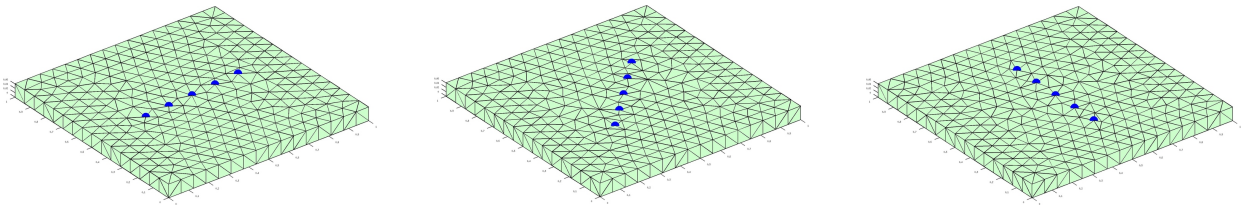
Next, we carry out the space-time convergence test for a range of spatial mesh sizes h and polynomial degrees p. Note that for all the numerical tests for our space-time method in this chapter, we use the same polynomial degree p for both space and time. Moreover, we set Δt (i.e. the thickness of each space-time mesh slab) equal to h in each case. For a comparison, we also solve the problem using the same space-time framework on a fixed mesh, where no vertices are rotated and thus the initial unstructured mesh remains unchanged for all time steps. In figure 6.2, a few sample space-time meshes are given to illustrate our space-time mesh slabs, and the bottom convergence plot compares the errors in the discrete space-time L^2-norm of our space-time DG method for the moving and the fixed mesh cases, respectively. It can be seen that the solutions from our moving meshes have essentially the same accuracy as the ones using a fixed mesh. Furthermore, from the convergence plot, the results clearly show that the orders of convergence are approximately $O(h^{p+1})$.

ALE Method with local L^2 Projection

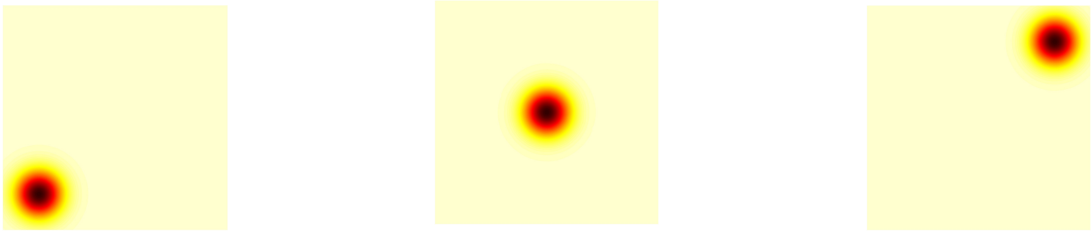
Alternatively, we simulate the same Euler vortex problem using our local ALE approach with frequent local L^2 projections to show its high-order accuracy.

First, we use the same moving mesh model as shown in figure 6.1 in order to generate large mesh deformations. The Euler equations 6.1 are then solved until time $T = 3.4\sqrt{2}$ but with timestep $\Delta t = 0.006$. We also reduce the angular velocity ω to ensure that the rotating nodes again rotate about 90 degrees within T. Note that since the novelty of this method is mainly in its spatial discretization and solution transferring techniques, we deliberately choose small time step Δt such that $\Delta t << h$ to ensure that the temporal numerical errors are negligible and truncation errors are dominated by the spatial discretization. This choice of small Δt is not due to the limitations of the edge flips or due to any stability restrictions from the discretization, we could have chosen a much larger Δt if we allowed for temporal errors, which has already been shown in the space-time convergence test.

We carry out the convergence test for a range of spatial mesh sizes h and polynomial degrees p, and compute inf-norms of the errors. Again, we solve the same problem using a



Mesh Slab of $\Omega[0, \Delta t]$ | Mesh Slab of $\Omega[\frac{T}{2} - \Delta t, \frac{T}{2}]$ | Mesh Slab of $\Omega[T - \Delta t, T]$



Pressure Field at $t = 0$ | Pressure Field at $t = \frac{T}{2}$ | Pressure Field at $t = T$

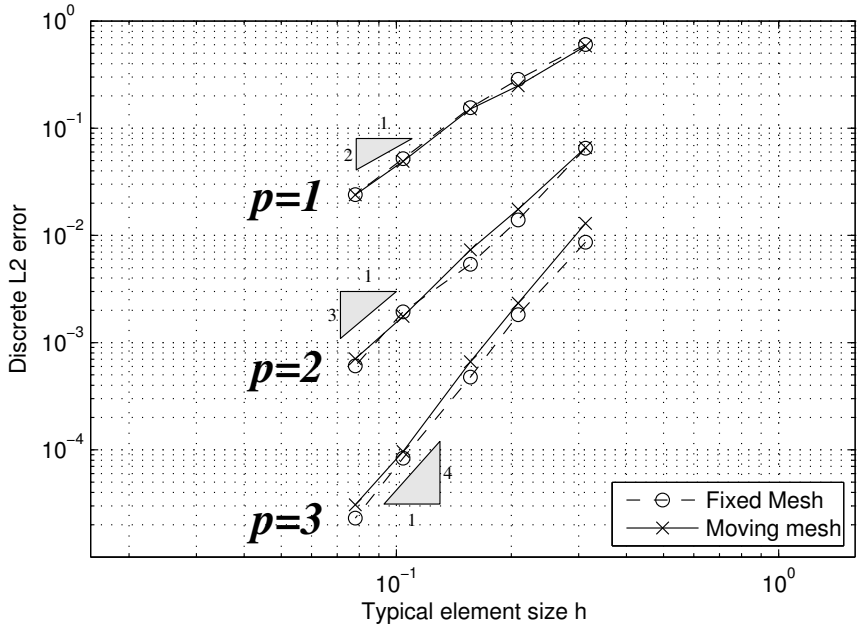


Figure 6.2: Space-Time Convergence Test for the Euler Vortex Problem. The top three plots are samples of space-time meshes with $\Delta t = h = 0.3125$ and $p = 3$. Below each mesh, the corresponding solutions are shown in pressure fields. The pressure plots uses the 'hot' colormap in Matlab with range $[68.5, 71.6]$. The bottom plot shows the convergence results for $p = 1$, 2 and 3.

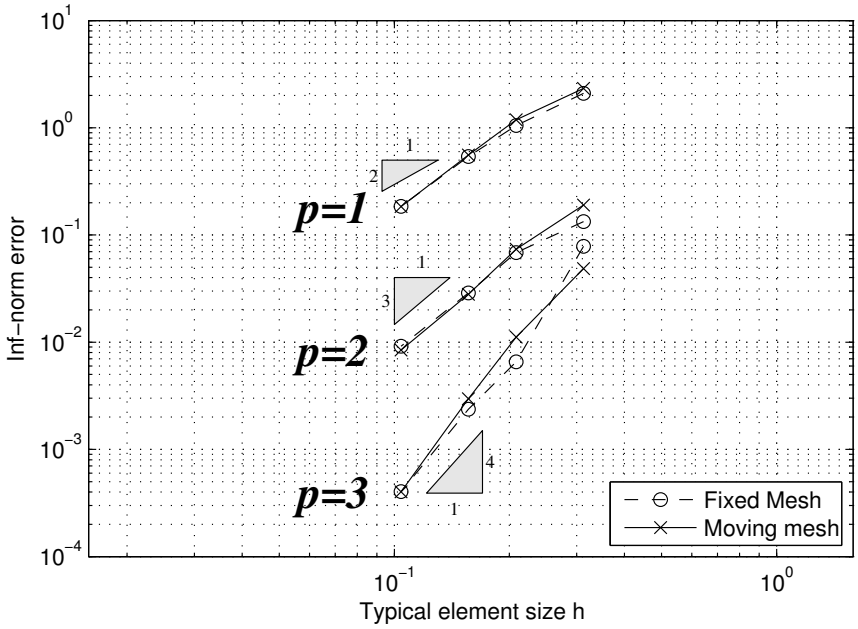


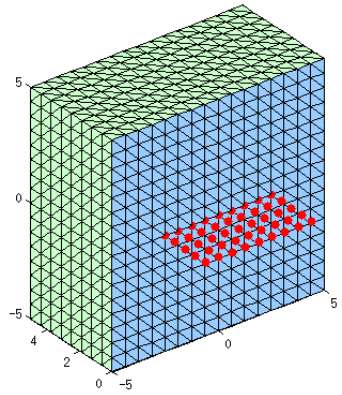
Figure 6.3: Euler-Vortex Convergence Plot for the ALE Method with Local L^2 Projections.

standard nodal DG scheme on the fixed initial mesh as a benchmark. From the resulting convergence plot in figure 6.3, it is still clear that for both methods, the optimal $O(h^{p+1})$ orders of convergence are obtained.

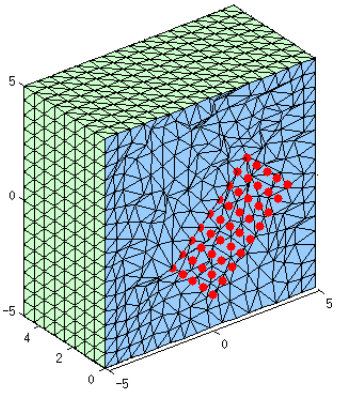
6.1.2 3D Case

The last Euler-vortex example is used to demonstrate the ability of our moving mesh strategy to adjust the mesh quality in 3D and achieve high-order accuracy for the ALE method with local L^2 projections. Similar to the 2D example, we propagate an Euler vortex cylinder in a $10 \times 10 \times 10$ cube (as shown in figure 6.4). The analytical solutions at (x, y, z, t) are the same as in Eqs 6.1 for each fixed time t. All the parameters are assigned he same values except that $(x_0, y_0) = (-2, -2)$, $M_\infty = 0.5$, $\epsilon = 3.0$ and $r_c = 0.75$.

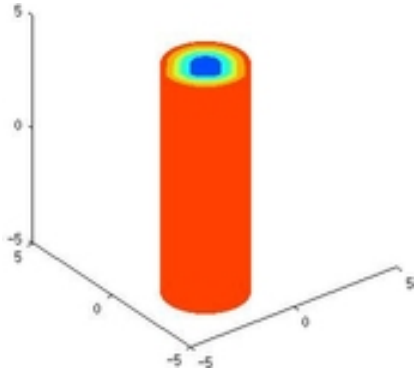
The mesh of the cube is initially generated as a 3D Cartesian grid. To show the effectiveness of our moving mesh approach, we induce large mesh deformations by rigidly rotating a set of interior mesh nodes with respect to the y-axis, which are showed in red in figure 6.4. These red nodes are chosen as all the mesh nodes of the initial grid with $x \in [-2.5, 2.5]$, $y \in [-2.5, 2.5]$ and $z = 0$. The Euler vortex travels from $t = 0$ to $t = T = \sqrt{3^2 + 3^2}$ and the set of red nodes are rotated about 30 degrees throughout the time interval $[0, T]$. With these rigid motions, the mesh quality decays rapidly and will eventually be unacceptable without mesh adjustments. On the contrary, when our moving mesh is employed, the mesh quality can be retained by smoothing mesh nodes and locally flipping the tetrahedra. In figure 6.4,



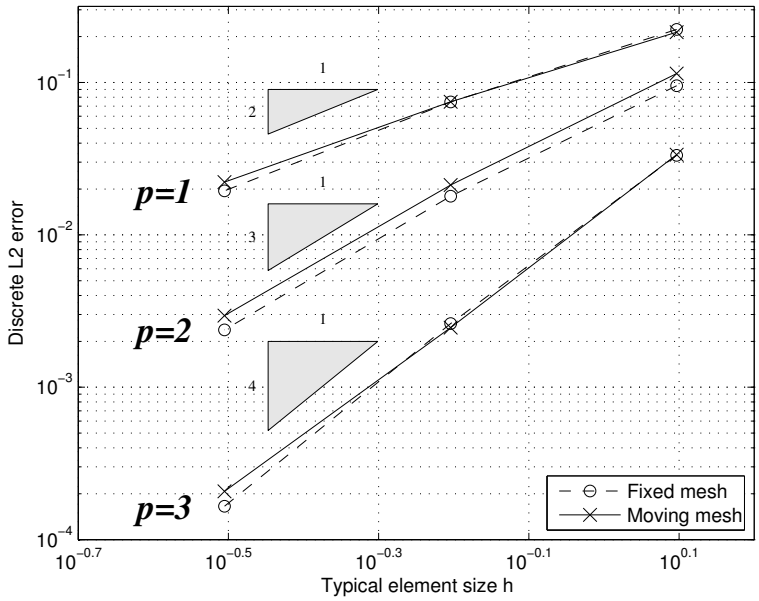
Moving Mesh at $t = 0$



Moving Mesh of $t = T$



Sample solution plot of Euler Vortex



Convergence Plot

Figure 6.4: Convergence Test for the 3D Euler Vortex Problem. The upper left and middle plots are two sample meshes at the initial and the final time, respectively. These meshes are generated on a $10 \times 10 \times 10$ cube. The blue faces show a cross-section of the tetrahedral mesh and the green faces are the outside surfaces of the cube. The red nodes are rotated rigidly. The right plot shows some sample pressure isosurfaces, which uses the 'jet' colormap in Matlab with range $[2.2377, 2.5871]$.

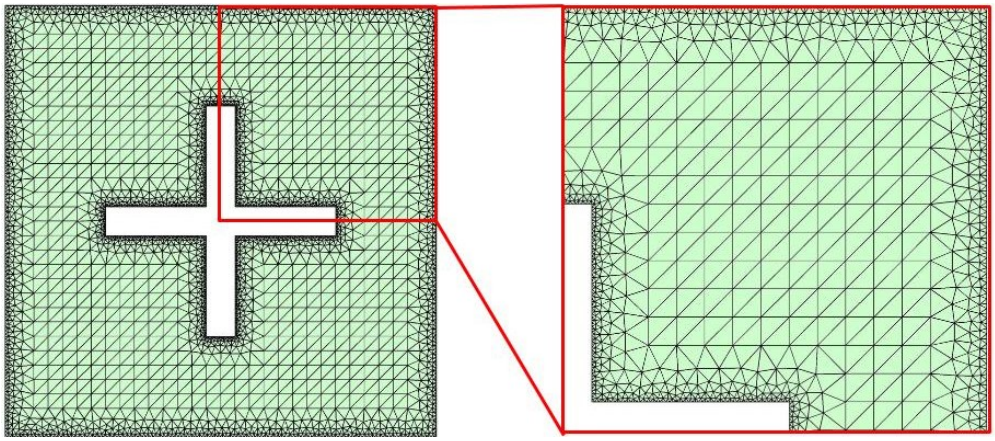


Figure 6.5: Initial Spatial Mesh of a Spinning Cross. The left zoom-in figure is for the upper right part of the initial mesh on the right.

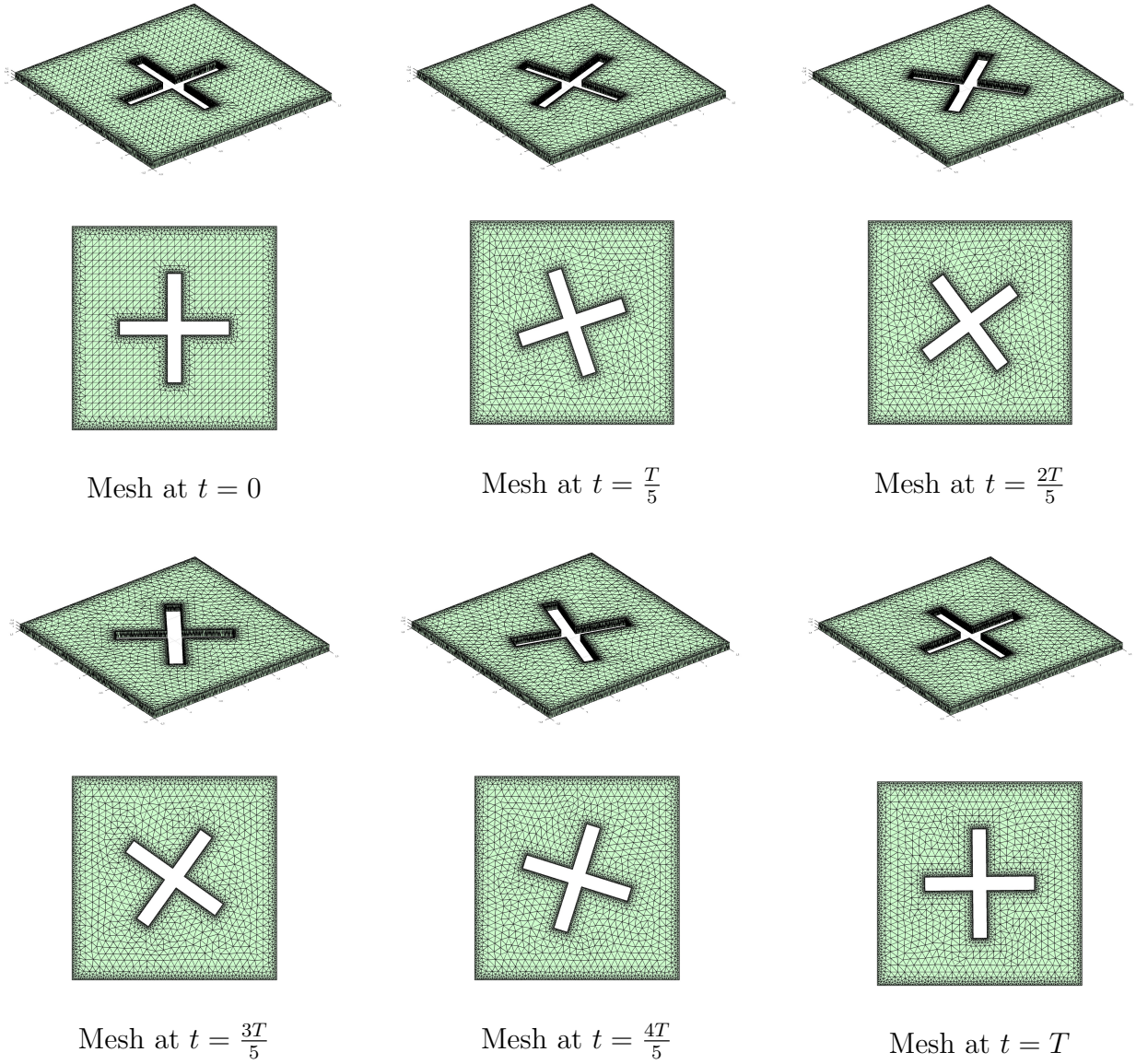
we can clearly see that the initially structured mesh quickly becomes fully unstructured and that it respects the motion of the red nodes. All element qualities remain acceptably high throughout the process.

Finally, we solve the 3D Euler vortex cylinder problem with periodic boundary conditions imposed on the vertical boundaries, using different spatial mesh sizes h and polynomial degrees p. Again, we choose $\Delta t << h$ so that the effect of the temporal error is negligable in this convergence test. As a benchmark, the same test is also carried out with the initial mesh fixed for all time steps, where the standard nodal DG scheme is implemented for the spatial discretization. The convergence plot in figure 6.4 illustrates that the test with moving meshes achieves the same optimal $O(h^{p+1})$ order of accuracy as the fixed mesh, even with frequent elements flipping in 3D.

6.2 Spinning Cross

Next, we consider a 3-by-3 square domain where a cross is spinning counterclockwise about the center, which is similar to the example problem in [65]. We solve the compressible Navier-Stokes equations starting from a zero-velocity initial condition with viscous wall conditions on all boundaries, and set the Mach number to 0.2 and the Reynolds number to 1500, based on the cross diameter 1.5 and the angular velocity $\omega = 1$.

As shown in figure 6.5, we initialize the mesh by gluing three parts together: two graded meshes around the fixed outside walls and the spinning cross walls, and a Cartesian grid in the middle. To simplify the mesh movement, we move the graded mesh around the cross rigidly with the geometry movement, fix the graded mesh around the outside walls, and only apply our moving mesh and flipping techniques to the uniform mesh in the middle.



Mesh at $t = 0$ | Mesh at $t = \frac{T}{5}$ | Mesh at $t = \frac{2T}{5}$

Mesh at $t = \frac{3T}{5}$ | Mesh at $t = \frac{4T}{5}$ | Mesh at $t = T$

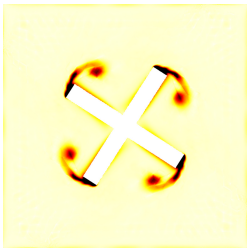
Figure 6.6: Examples of Spatial Meshes and Space-Time Mesh Slabs. Note that to better illustrate our space-time mesh structure, we rescale the thickness of our space-time mesh slabs, which is not equal to the real Δt we used for the simulation.

Figure 6.6 shows that our method can retain the quality of the elements even with the large deformations induced by the spinning cross.

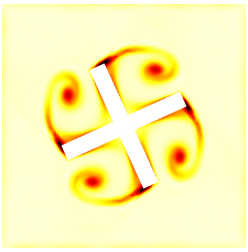
We set $\Delta t = 0.01$ and simulate the compressible flow around this spinning cross until $T = 3\pi$. Again, we implement both of our space-time DG and ALE-DG schemes for the same problem.

For the space-time DG scheme, a few space-time mesh slabs are given in figure 6.6. In

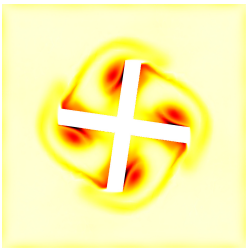
order to better resolve the solution, we implement our method with polynomial orders $p = 2$ and linear element geometries. Entropy plots of the flow solutions at a few time steps are shown in figure 6.7. Note that the optimal order of accuracy $O(h^{p+1})$ will not be achieved unless the boundary of the cross is also curved, which is an important future improvement.



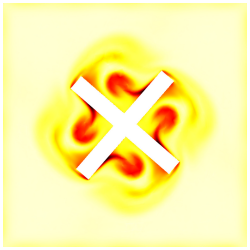
Entropy Plot at $t = 1.0$



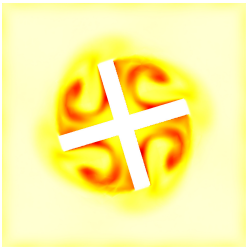
Entropy Plot at $t = 2.0$



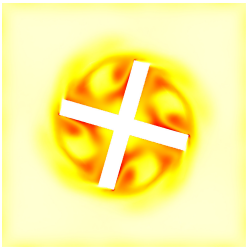
Entropy Plot at $t = 3.0$



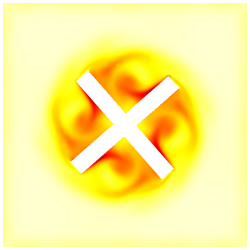
Entropy Plot at $t = 4.0$



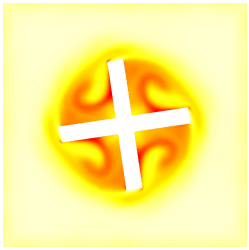
Entropy Plot at $t = 5.0$



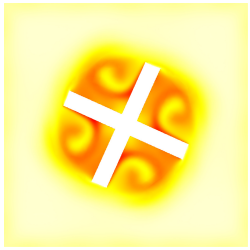
Entropy Plot at $t = 6.0$



Entropy Plot at $t = 7.0$



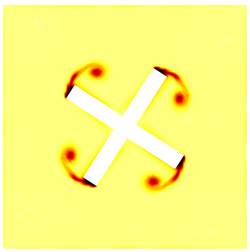
Entropy Plot at $t = 8.0$

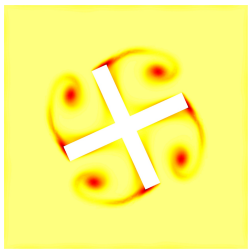


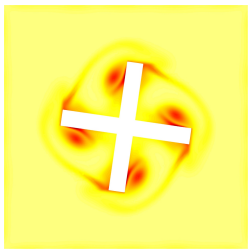
Entropy Plot at $t = 9.0$

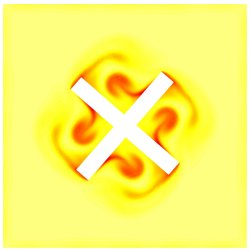
Figure 6.7: Space-Time DG solutions for Compressible Navier-Stokes flow in a domain with a spinning cross (entropy). It uses the reverse 'hot' colormap in Matlab with range $[17.85, 18.05]$.

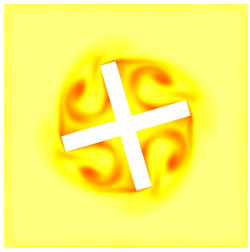
For our ALE DG method, we use polynomial degrees $p = 3$. Entropy plots of the flow solutions at a few time steps are shown in figure 6.8. From these entropy plots we can see that our local L^2 projections keep the solutions from losing the numerical accuracy, even with frequent mesh topology changes based on our moving-mesh strategy.

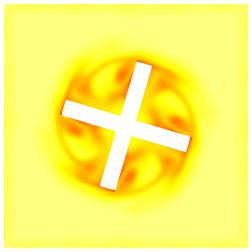

Entropy Plot at $t = 1.0$

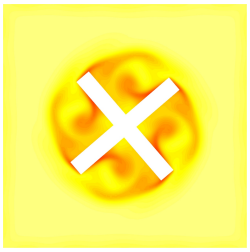

Entropy Plot at $t = 2.0$

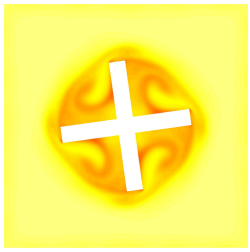

Entropy Plot at $t = 3.0$


Entropy Plot at $t = 4.0$


Entropy Plot at $t = 5.0$


Entropy Plot at $t = 6.0$


Entropy Plot at $t = 7.0$


Entropy Plot at $t = 8.0$

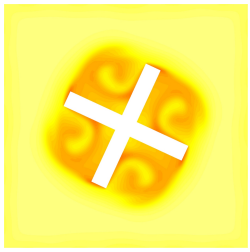

Entropy Plot at $t = 9.0$

Figure 6.8: ALE DG solutions for Compressible Navier-Stokes flow in a domain with a spinning cross (entropy). It uses the reverse 'hot' colormap in Matlab with range $[18.02, 18.32]$.

6.3 Pitching Tandem Airfoils

Next, we consider a compressible Navier-Stokes simulation similar to the one studied in [85]. It consists of two pitching NACA0012 airfoils with chord length $c = 1$ in a rectangular domain. As shown in figure 6.9, at the initial time $t = 0$ the two foils have a zero pitching angle and are aligned on the horizontal axis close to each other. The distance between the trailing edge of the first foil and the leading edge of the second foil is $d = 0.1$. The two foils are both treated as rigid bodies and rotated around the points $p = c/3$ to the right of their leading edges. The rotation follows a prescribed harmonic function as

$$\theta = A \sin(-2\pi f t) \tag{6.2}$$

where $A = \pi/6$ and $f = 0.05$. The flow has Mach number 0.2 and Reynolds number 3000.

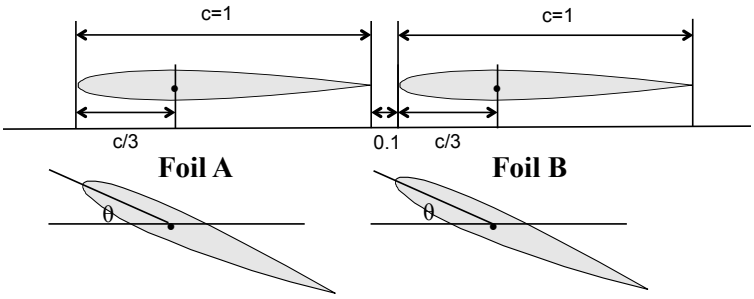
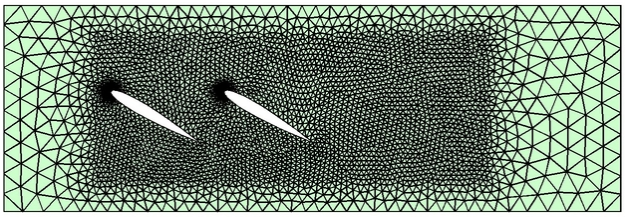
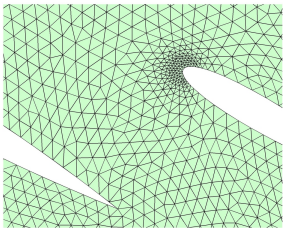


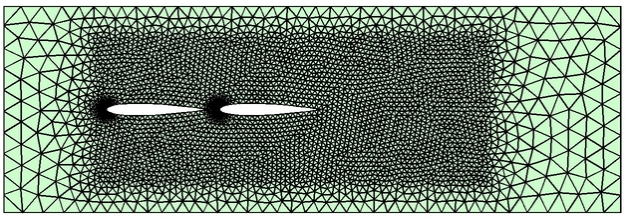
Figure 6.9: Schematics of the pitching tandem airfoil problem.



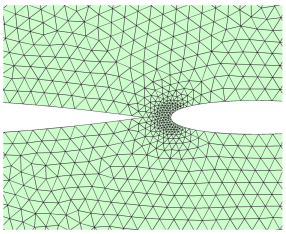
Unstructured Mesh of the spatial domain at $t = 5.0$



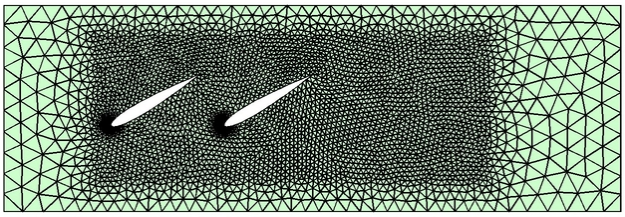
Zoom-in Mesh at $t = 5.0$



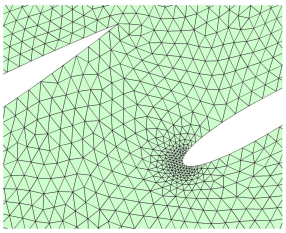
Unstructured Mesh of the spatial domain at $t = 10.0$



Zoom-in Mesh at $t = 10.0$



Unstructured Mesh of the spatial domain at $t = 15.0$



Zoom-in Mesh at $t = 15.0$

Figure 6.10: Sample Spatial Meshes of Two Pitching Tandem Airfoils.

As the two foils are placed very close and rotated based on the same harmonic function, the mesh quality decays very rapidly if only the position of the nodes are updated. On the contrary, our moving mesh strategy only requires an unstructured two-dimensional mesh of the initial domain Ω^0 and is able to improve the mesh automatically by local mesh operations. In this way, the moving mesh remains well shaped and avoids inverted elements. The mesh motion is illustrated in figure 6.10.

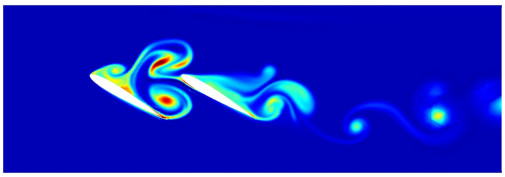
We implement both our high-order space-time DG method and ALE-DG method with polynomial degrees $p = 2$ and linear element geometries. The compressible Navier-Stokes equations are solved with viscous wall conditions on the airfoils and far-field conditions on the outside boundaries. A couple of sample entropy plots are shown in figure 6.11. In figure 6.12, drag and lift coefficients computed from these two methods are compared. Through the plots, although the simulation of these unsteady flows is highly sensitive to small perturbations, we can conclude that the two approaches are able to predict similar forces.

6.4 Airfoil with a Deploying Spoiler

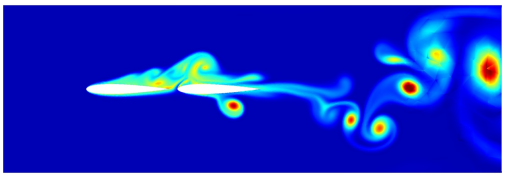
As an example of a more complicated domain deformation, we use our space-time DG method to solve for the compressible flow around a NACA0012 airfoil with chord length 1, in a 6×2 rectangular domain, similar to the problem introduced in [64]. As illustrated in figure 6.13, the foil is located between $x = 0$ to $x = 1$ with axis of symmetry $y = 0$. We then remove a right triangle with curved hypotenuse from $x = 0.6383$ to $x = 0.7534$ and replace it by a thin spoiler of length 0.1. We keep a horizontal gap of width 2×10^{-3} between the foil and the spoiler, which are only connected at the point $(0.6383, 0.0422)$. An adaptive mesh is applied with refined elements around the spoiler. When the spoiler is deployed, it rotates about the connecting point with the foil with the angular velocity 0.1, which generates a large domain deformation around the spoiler. To address this, we update the adaptive mesh size function at each timestep and improve the mesh quality by our local mesh operations.

We impose viscous wall conditions on both the airfoil and the spoiler, and far-field conditions on the outside boundaries. The numerical simulation starts with a steady flow around the airfoil with a closed spoiler, at Mach number 0.2 and Reynolds number 5000, based on the airfoil chord length 1 and the free-stream velocity 1. Next, we fix the foil but raise the spoiler gradually up to a 90 degrees angle, which results in massive flow separation behind the foil. We keep the spoiler at the vertical state for a short time period, and then close it again by reversing the motion. During this entire process, we use our space-time DG method to solve for the compressible viscous flow during the raising and the closing part, and a regular two-dimensional method-of-lines DG method for the time period when the spoiler position is fixed. Again, as in the previous tests, we use polynomial orders $p = 2$ with linear element geometries, in order to better resolve the solution fields.

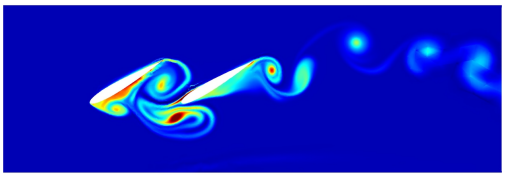
In figure 6.13, some mesh plots are given to show how our local mesh operations improve the spatial mesh as the spoiler is raised, and three samples of entropy plots are shown in figure 6.14. In the zoom-in plots, we can confirm that our space-time DG method retains the



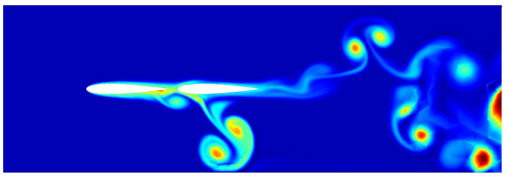
Entropy Plot at $t = 5.0$ (Space-Time)



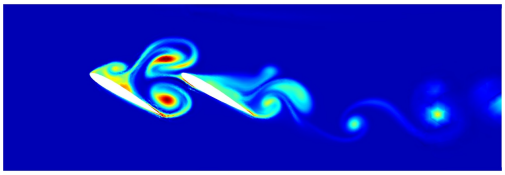
Entropy Plot at $t = 10.0$ (Space-Time)



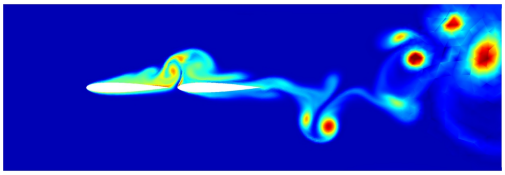
Entropy Plot at $t = 15.0$ (Space-Time)



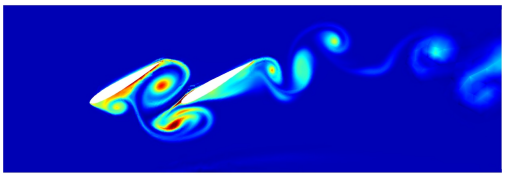
Entropy Plot at $t = 20.0$ (Space-Time)



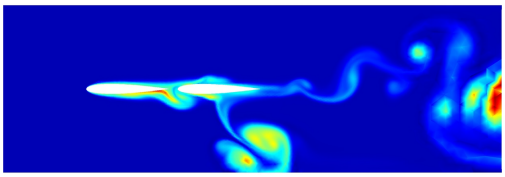
Entropy Plot at $t = 5.0$ (ALE)



Entropy Plot at $t = 10.0$ (ALE)



Entropy Plot at $t = 15.0$ (ALE)



Entropy Plot at $t = 20.0$ (ALE)

Figure 6.11: Numerical results for the pitching tandem airfoils. There are solution fields for both space-time and ALE methods (entropy of the flow at 4 time instances). It uses the 'jet' colormap in Matlab with range $[17.79, 18.18]$.

high quality of the solutions even for the large deformation between the foil and the spoiler. The lift and the drag coefficients during the entire process are shown in figure 6.15.

6.5 Double Vertical Axis Wind Turbines

Inspired by section 6.2, we can see that our numerical schemes are particular useful for solving problem with rotating geometries. Therefore, in this section, we will apply our ALE-DG method to simulate the 2D compressible flow around vertical axis wind turbines (VAWTs). Here we mainly discuss the numerical methods used in this application, and more details about the engineering designs and the physical interpretations can be found in [38].

Recent studies show that the vertical axis wind turbines have some advantages over hori-

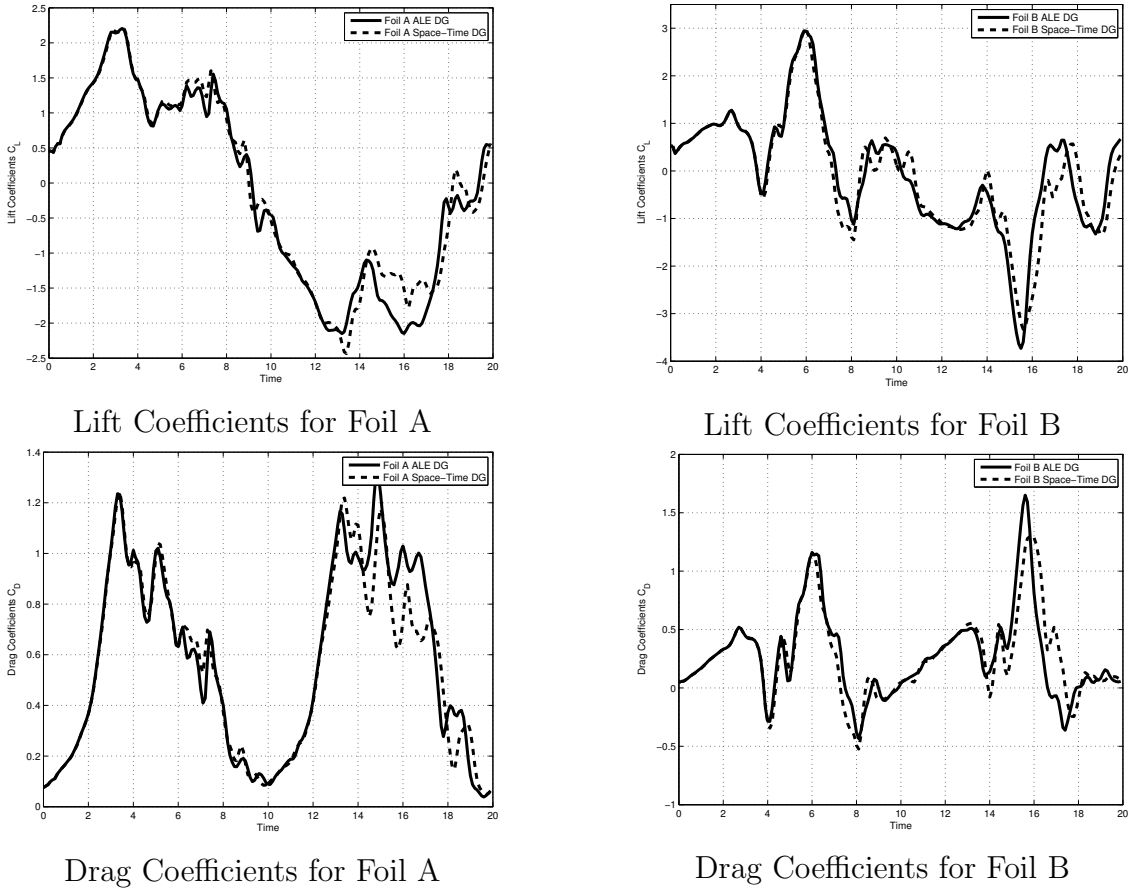
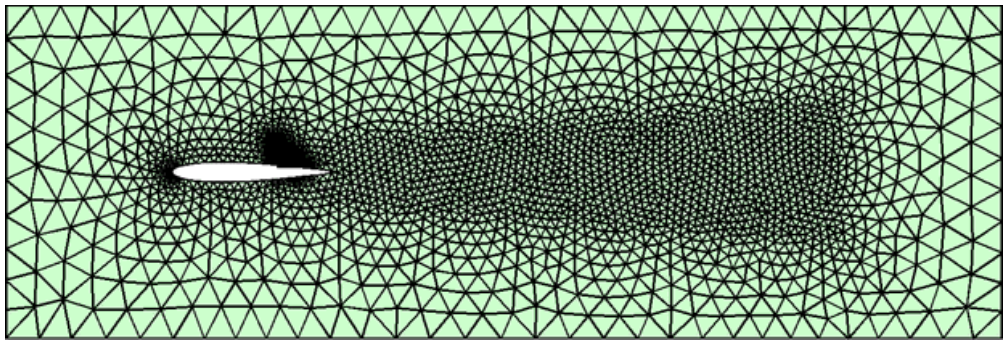


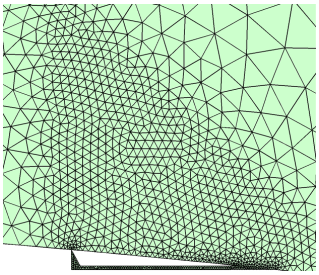
Figure 6.12: The drag and lift forces on the pitching tandem NACA0012 airfoils as a function of time for both the space-time and the ALE methods.

zontal axis wind turbines such as their ability to produce electricity in any wind direction as well as its lower production and maintenance cost. Similar to many engineering problems, the design of VAWTs usually requires considerable experimental costs, and computational approaches are often employed to provide high-fidelity simulations in order to better understand the mechanism and optimize the VAWT design. In this section, we simulate the compressible isentropic flow using Large Eddy Simulation (LES). As LES performs best for low to medium Reynolds number simulations, we model the turbines in conditions similar to if the blades were rotating in water, such as the turbine studied experimentally in [73].

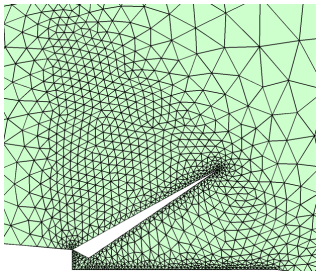
In [38], the simulation of 2D isentropic turbulent flow around a single VAWT is first studied. As for the numerical method, it is quite straightforward to apply the standard ALE framework with a nodal DG scheme (see [57]) to this problem, and a smooth ALE mapping can be simply constructed by rotating the entire mesh. Then the problem moves to the simulation related to multiple VAWTs, or a so-called 'wind farm'. This problem has substantial practical significance since experimental results show that the use of counter-



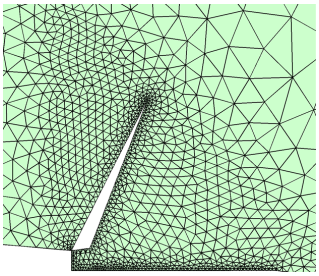
The initial spatial mesh at $t = 0.0$



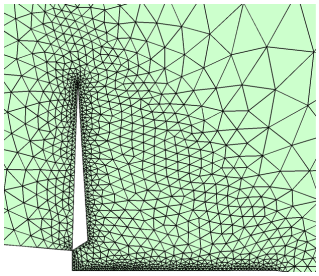
Zoom-in mesh near spoiler at $t = 0.0$



Zoom-in mesh near spoiler at $t = 6.0$



Zoom-in mesh near spoiler at $t = 12.0$

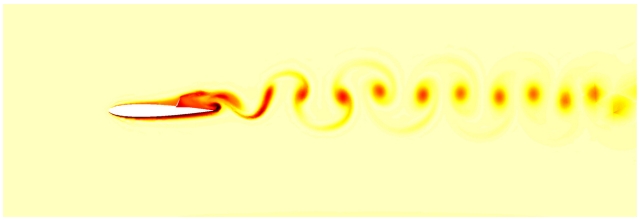


Zoom-in mesh near spoiler at $t = 18.0$

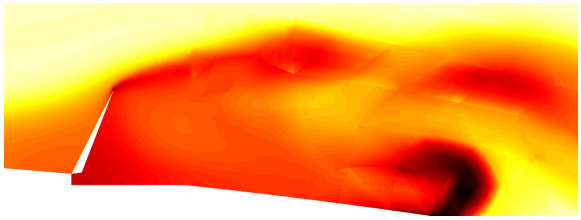
Figure 6.13: Spatial Meshes of the Airfoil with a Deploying Spoiler.

rotating VAWTs is able to generate higher power output per unit land area [16]. Here we simplify the model to a double counter-rotating VAWT, as illustrated in figure 6.16.

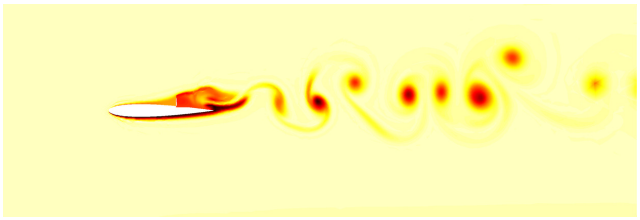
However, unlike the single VAWT model, the double counter-rotating VAWTs are more difficult to simulate directly using the ALE framework. The entire mesh cannot be rotated like in the single turbine case, and due to the counter-rotating motion it is almost impossible to find a smooth mapping with unchanged mesh topology, in particular for the area in-between the turbines. Instead, we solve this problem using our ALE-DG method with local L^2 projections, which can easily handle arbitrary mesh topology changes.



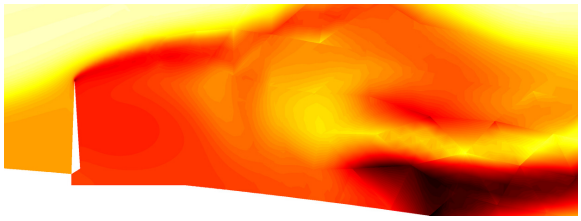
Entropy Plot at $t = 12.0$



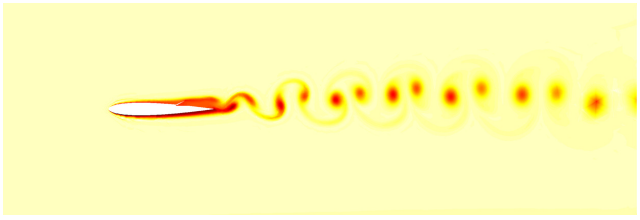
Zoom-in Entropy Plot near spoiler at $t = 12.0$



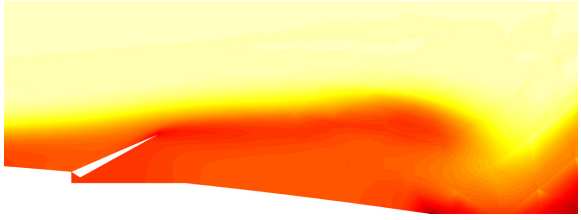
Entropy Plot at $t = 28.0$



Zoom-in Entropy Plot near spoiler at $t = 28.0$



Entropy Plot at $t = 44.0$



Zoom-in Entropy Plot near spoiler at $t = 44.0$

Figure 6.14: Space-Time DG Solutions for Compressible Navier-Stokes flow around an airfoil with a deploying spoiler. It uses the reverse 'hot' colormap in Matlab with range $[17.79, 18.18]$.

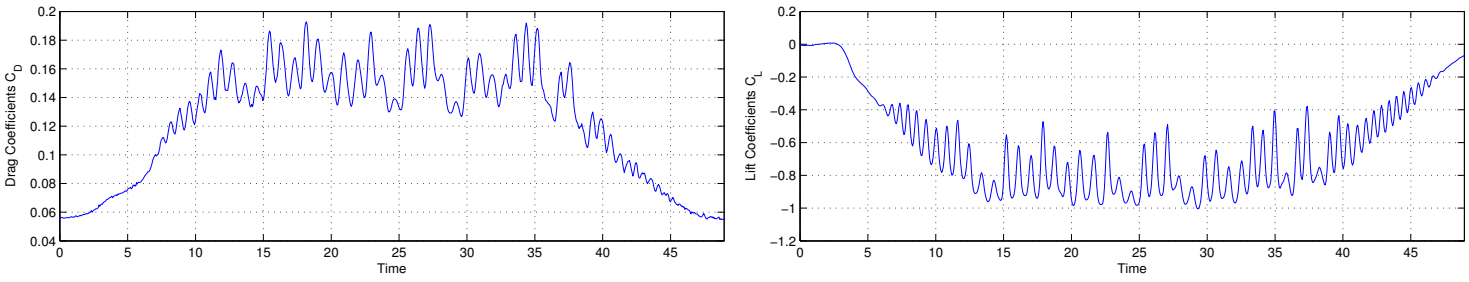


Figure 6.15: Drag and lift coefficients around the NACA0012 airfoil with a deploying spoiler as a function of time.

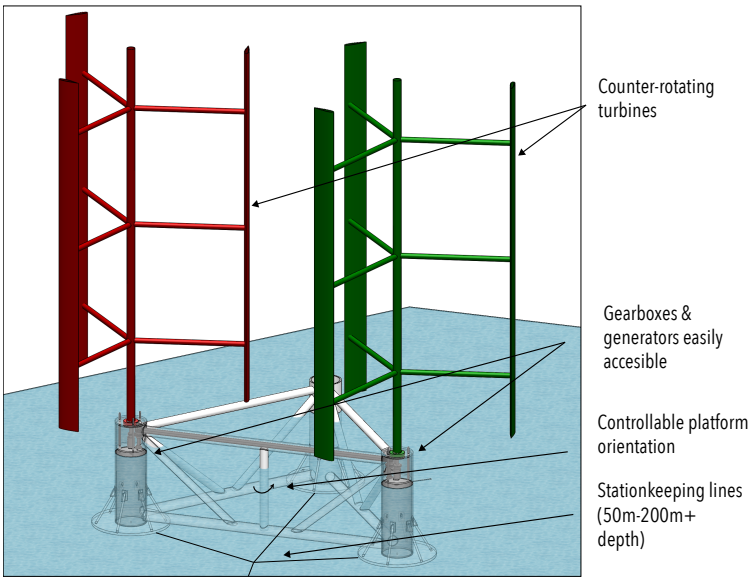
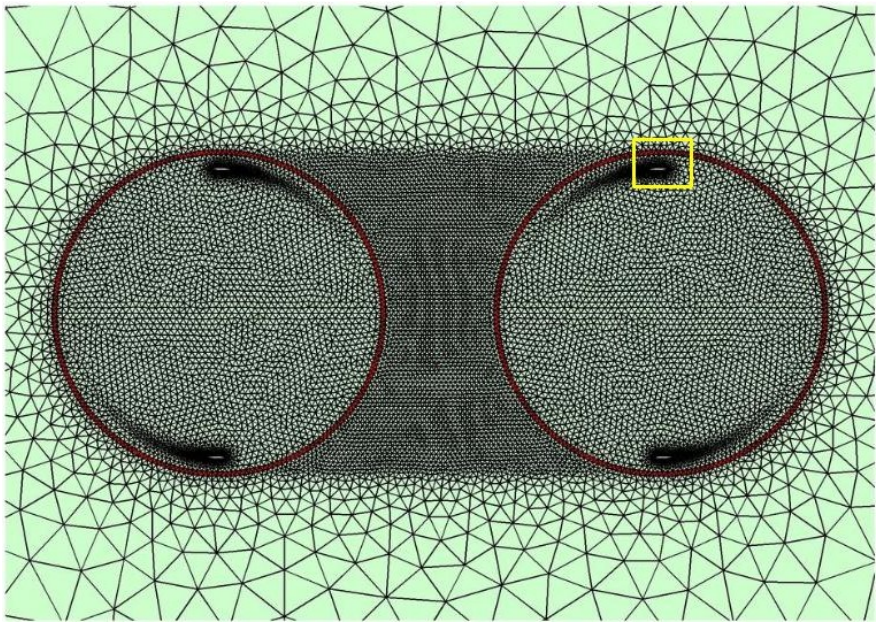


Figure 6.16: Schematics of the double vertical axis wind turbines. (Image from [38])

Our model is defined on an 8-by-4 meter rectangular domain and we use two counter-rotating turbines, similar to the single turbine built in [73]. This turbine has two NACA0012 airfoils with chord lengths $c = 9.14$ m as the blades for each turbine. The radius of each turbine is $r = 0.61$ m and the distance between the centers of the turbines is $d = 1.5r$. The left turbine is rotating counter-clockwise with angular velocity $\omega \approx 0.75$ rad/s and the right one is rotating clockwise with the same magnitude of angular velocity. The free-stream velocity is $[0, -1]^T$ m/s which means that the wind goes from north to south. The Mach number $M = 0.02$ and the kinematic viscosity $\nu = 10^{-6}$ m/s^2.

As shown in figure 6.17, we create a mesh for the two turbines and for the outside area, respectively. We then glue these three parts of the mesh together by connecting the boundary nodes between the mesh of the outside area to that of turbines. This form two layers of elements around the two turbines, whose elements are colored in red in figure 6.17. To enhance the efficiency of our moving-mesh strategy, we restrict our edge flipping operations to only these two layers of triangles. This mesh motion is also illustrated in figure 6.17.

We then solve the compressible isentropic flow around these two counter-rotating turbines with timestep $\Delta t = 0.005$ s and polynomial degrees $p = 3$. Again, viscous wall conditions are imposed on the turbine blades and far-field conditions on the outside boundaries. The simulation results are plotted in figure 6.18 in different physical quantities. Through the plots, we can see that even with frequent L^2 projections around the turbines, our ALE-DG method is able to resolve the solutions as the turbines spin, and capture most of the features of the turbulent flow around blades.



The initial spatial mesh at $t = 0.0$

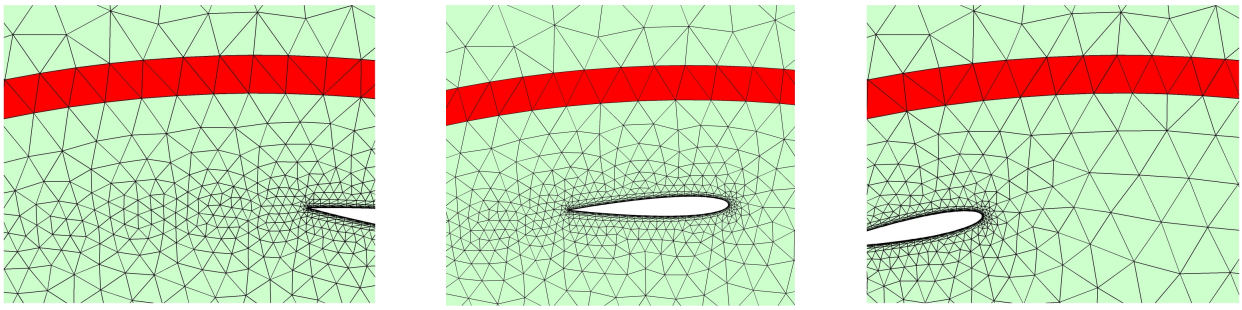
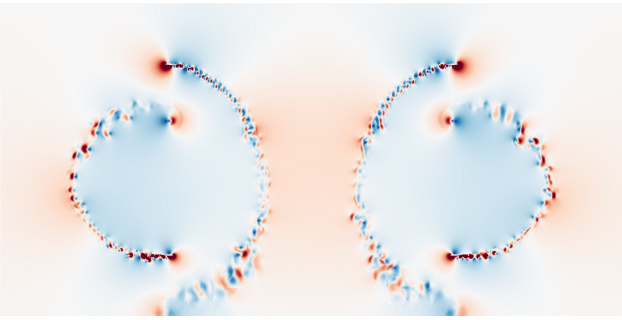
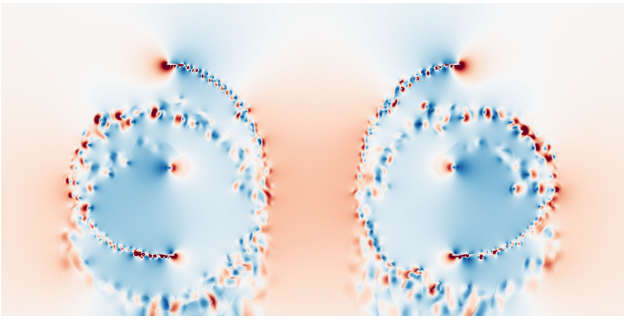


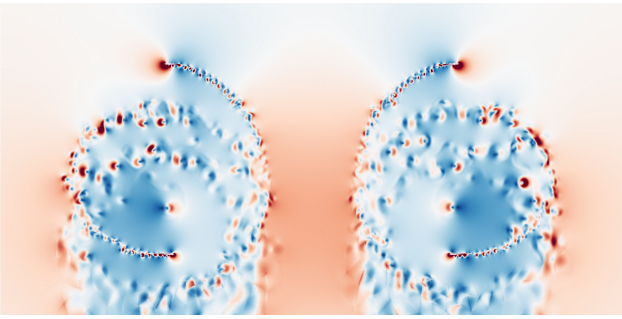
Figure 6.17: Unstructured Mesh for Double VAWTs. The initial mesh is showed on the top, where all the edge flipping operations happen in the area colored in red. In order to show the mesh motion, three zoom-in plots are placed on the bottom for the area circled by a yellow window in the top plot.



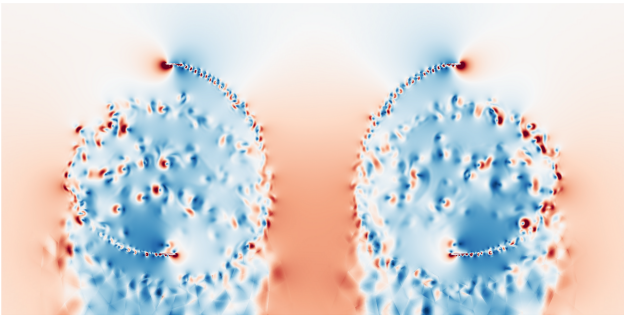
u_2 Plot at $t = 0.84$



u_2 Plot at $t = 1.68$

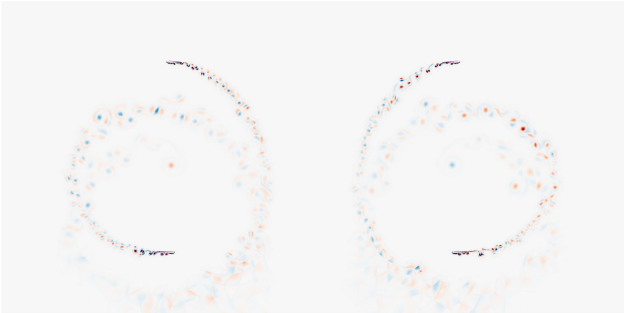


u_2 Plot at $t = 2.52$

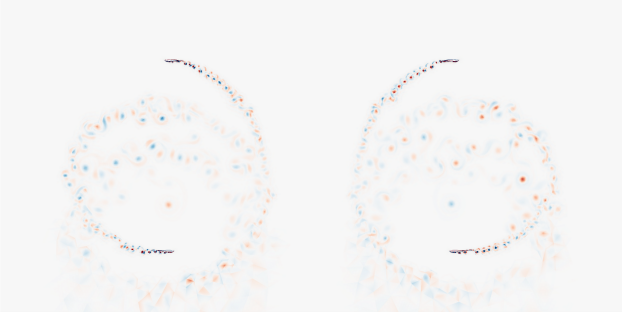


u_2 Plot at $t = 3.36$

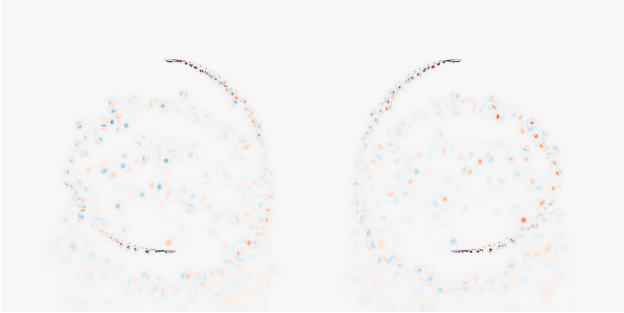
Vorticity Plot at $t = 0.84$



Vorticity Plot at $t = 1.68$

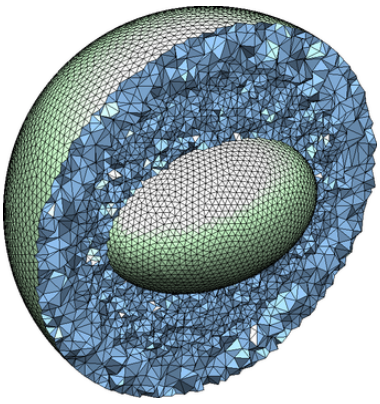


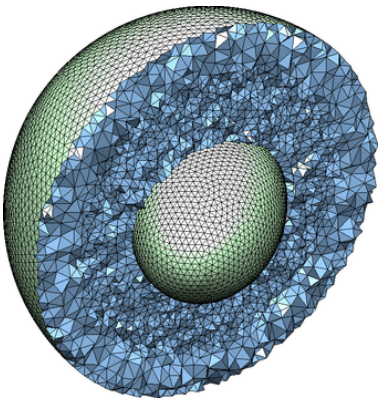
Vorticity Plot at $t = 2.52$

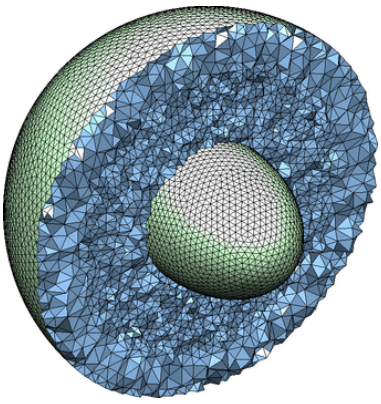


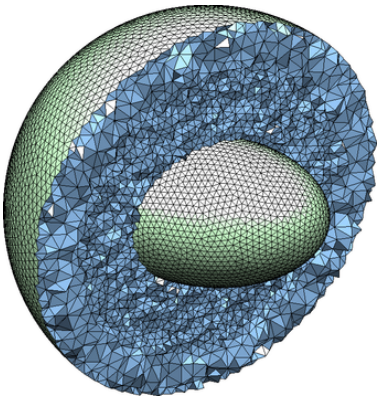
Vorticity Plot at $t = 3.36$

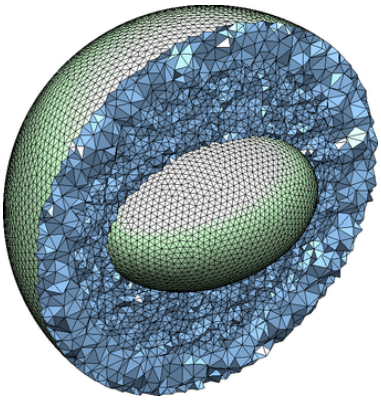
Figure 6.18: Numerical results for double VAWTs. Sample solutions are plotted for the y-component of the velocity and for the vorticity field, respectively. The plots for y-component of the velocity used reverse RdBu colormap in Python's matplotlib module. The color range is $[0.03656, 0.14624]$. The plots for vorticity used RdBu colormap with the color range is $[-50, 50]$.

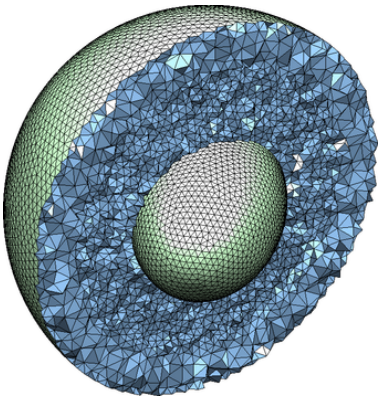

Mesh at $t = 0.0$

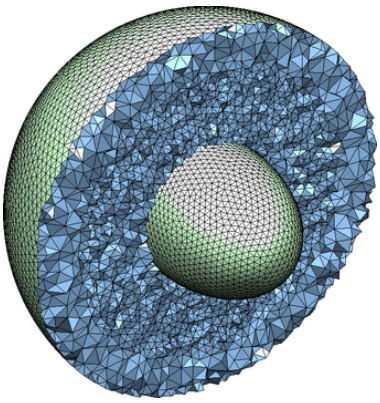

Mesh at $t = 2.5$

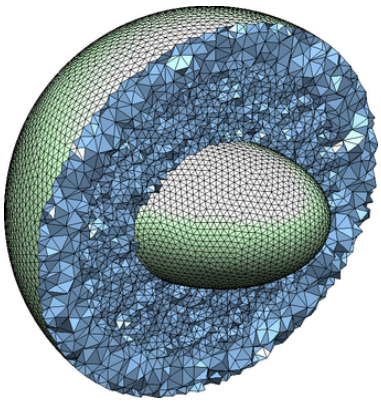

Mesh at $t = 5.0$


Mesh at $t = 7.5$


Mesh at $t = 10.0$


Mesh at $t = 12.5$


Mesh at $t = 15.0$


Mesh at $t = 17.5$

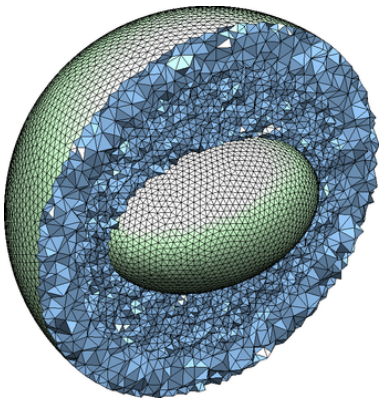

Mesh at $t = 20.0$

Figure 6.19: Sample 3D Meshes for Spinning Ellipsoid Problem. The surface mesh of the inside ellipsoid and outside sphere is colored in green. The blue faces show a cross-section of the spherical mesh.

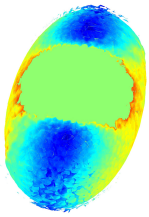
6.6 Rotating 3D Ellipsoid

Next, we demonstrate that our ALE-DG method has the capability to handle large deformation problems in 3D. Here our test case is analog to the section 6.2, and we consider a spherical domain with a spinning ellipsoid in the center. The radius of the outside sphere is 10 and the centered ellipsoid has three semi-principal axes of length 6, 3.75 and 3.75, and rotates about the z-axis with angular velocity $\omega = 0.1\pi$rad/s.

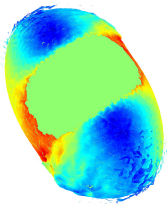
As shown in figure 6.19, we use an adaptive tetrahedral mesh to refine the area around the ellipsoidal and the spherical boundaries in order to resolve the solution. As the ellipsoid spins, our moving-mesh strategy keeps adjusting the mesh quality by locally changing the mesh connectivity, and avoids the appearance of inverted elements.

The compressible flow starts from zero free-stream velocity, and has Mach number 0.2 and dynamical viscosity $\mu = 1/1500$. The viscous wall conditions are set for both the central spinning ellipsoid and the outside sphere. We then use our ALE-DG method to solve the flow problem with polynomial degrees $p = 2$ and locally transfer the solution during mesh topology changes using our local L^2 projections (figure 5.6). Finally, some sample solutions are illustrated in figure 6.20.

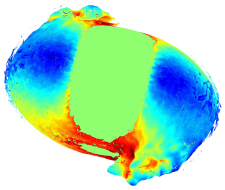
Solution at $t = 0.0$



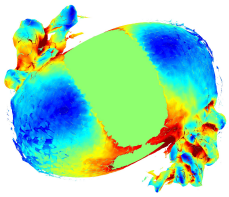
Solution at $t = 2.5$



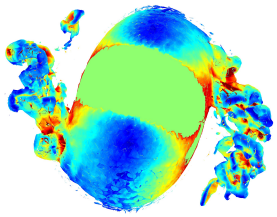
Solution at $t = 5.0$



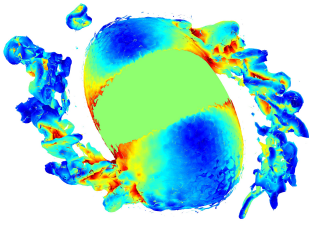
Solution at $t = 7.5$



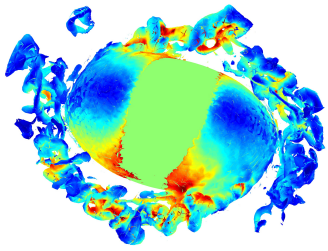
Solution at $t = 10.0$



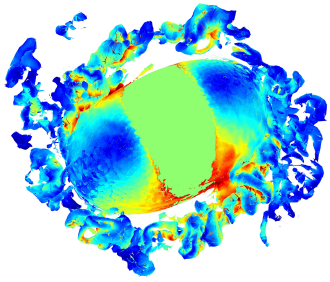
Solution at $t = 12.5$



Solution at $t = 15.0$



Solution at $t = 17.5$



Solution at $t = 20.0$

Figure 6.20: Sample Solutions for Spinning Ellipsoid Problem. The surface of the spinning ellipsoid is painted in green and we plot the Mach solution on the entropy isosurface. It uses the 'jet' colormap in Matlab with range $[0, 0.25]$.

Chapter 7

Conclusion & Future Work

In this thesis, we proposed new methods for solving compressible flow problems on moving domains with large deformations. First, we described our moving-mesh strategy which employs a number of local mesh operations to adjust for arbitrary geometric motion. Based on this strategy, we developed two different numerical methods for this class of problems. The first approach derived a DG scheme in a space-time framework and introduced efficient and robust algorithms for generating space-time meshes. The second approach considered the DG scheme in an ALE framework and handled the solution transferring for mesh topology changes using efficient local L^2 projections. Finally, we verified the high-order accuracy and the capability to address large deformation problems of our two numerical methods using various numerical simulations.

For future work, since currently we are mainly working on linear elements, the extension to curved meshes is critical in order to achieve high-order accuracy in problems with curved boundaries. This involves both the curved space-time mesh generation and the implementation of the local L^2 projections for curved elements. Moreover, the robustness of our moving-mesh strategy and the resulting mesh quality should be investigated, in particular for the 3D case. To improve the robustness, we should be able to dynamically pick the pseudo time-step in the DistMesh algorithm in a more systematic way and to add some additional local mesh topology operations to handle more configurations of tetrahedra. Also, since problems with complicated motions may include topology changes, we also need to handle this difficulty with some special treatment around the domain boundary. Computationally, for large-scale simulations, it is necessary to parallelize both our space-time mesh generation and the local L^2 projections, which is non-trivial and requires some techniques to communicate solutions and meshes between different processes. Finally, like for the VAWT simulation, more real engineering applications should be carried out for both our methods, which will help us better understand these methods and to improve them in order to meet the specified requirements arising from different needs.

Bibliography

[1] Reza Abedi et al. "Spacetime meshing with adaptive coarsening and refinement". In: *4th Symposium on Trends in Unstructured Mesh Generation, 7th US National Congress on Computational Mechanics*. Citeseer. 2003.

[2] F. Alauzet. "Efficient moving mesh technique using generalized swapping". In: *Proceedings of the 21th International Meshing Roundtable*. Sandia Nat. Lab., 2012, pp. 17–37.

[3] SK Aliabadi and TE Tezduyar. "Space-time finite element computation of compressible flows involving moving boundaries and interfaces". In: *Computer Methods in Applied Mechanics and Engineering* 107.1 (1993), pp. 209–223.

[4] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. "Unified analysis of discontinuous Galerkin methods for elliptic problems". In: *SIAM J. Numer. Anal.* 39.5 (2001/02), pp. 1749–1779. ISSN: 1095-7170.

[5] Farinatti Aymone and José Lufs. "Mesh motion techniques for the ALE formulation in 3D large deformation problems". In: *International journal for numerical methods in engineering* 59.14 (2004), pp. 1879–1908.

[6] Antonio Baeza and Pep Mulet. "Adaptive mesh refinement techniques for high-order shock capturing schemes for multi-dimensional hydrodynamic simulations". In: *International journal for numerical methods in fluids* 52.4 (2006), pp. 455–471.

[7] Francesco Bassi and Stefano Rebay. "A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier–Stokes equations". In: *Journal of computational physics* 131.2 (1997), pp. 267–279.

[8] Marek Behr. "Simplex space–time meshes in finite element simulations". In: *International journal for numerical methods in fluids* 57.9 (2008), pp. 1421–1434.

[9] Henning Braess and Peter Wriggers. "Arbitrary Lagrangian Eulerian finite element analysis of free surface flow". In: *Computer Methods in Applied Mechanics and Engineering* 190.1 (2000), pp. 95–109.

[10] Paul Castillo. "Performance of discontinuous Galerkin methods for elliptic PDEs". In: *SIAM Journal on Scientific Computing* 24.2 (2002), pp. 524–547.

[11] Alexandre Joel Chorin, Jerrold E Marsden, and Jerrold E Marsden. *A mathematical introduction to fluid mechanics*. Vol. 3. Springer, 1990.

[12] B. Cockburn and C.-W. Shu. "Runge-Kutta discontinuous Galerkin methods for convection-dominated problems". In: *J. Sci. Comput.* 16.3 (2001), pp. 173–261. ISSN: 0885-7474.

[13] Bernardo Cockburn and Chi-Wang Shu. "The local discontinuous Galerkin method for time-dependent convection-diffusion systems". In: *SIAM Journal on Numerical Analysis* 35.6 (1998), pp. 2440–2463.

[14] Phillip Colella, Daniel T. Graves, Benjamin J. Keen, and David Modiano. "A Cartesian grid embedded boundary method for hyperbolic conservation laws". In: *J. Comput. Phys.* 211.1 (2006), pp. 347–366. ISSN: 0021-9991.

[15] Gaëtan Compere, Jean-François Remacle, Johan Jansson, and Johan Hoffman. "A mesh adaptation framework for dealing with large deforming meshes". In: *International journal for numerical methods in engineering* 82.7 (2010), pp. 843–867.

[16] John O Dabiri. "Potential order-of-magnitude enhancement of wind farm power density via counter-rotating vertical-axis wind turbine arrays". In: *Journal of Renewable and Sustainable Energy* 3.4 (2011), p. 043104.

[17] A De Boer, MS Van der Schoot, and H Bijl. "Mesh deformation based on radial basis function interpolation". In: *Computers & structures* 85.11 (2007), pp. 784–795.

[18] James W Demmel. *Applied numerical linear algebra*. Siam, 1997.

[19] Benoıt Desjardins, Emmanuel Grenier, P-L Lions, and Nader Masmoudi. "Incompressible Limit for Solutions of the Isentropic Navier–Stokes Equations with Dirichlet Boundary Conditions". In: *Journal de mathématiques pures et appliquées* 78.5 (1999), pp. 461–471.

[20] VA Dobrev, TE Ellis, Tz V Kolev, and RN Rieben. "Curvilinear finite elements for Lagrangian hydrodynamics". In: *International Journal for Numerical Methods in Fluids* 65.11-12 (2011), pp. 1295–1310.

[21] Jean Donea. "Arbitrary Lagrangian-Eulerian finite element methods". In: *Computational methods for transient analysis(A 84-29160 12-64). Amsterdam, North-Holland, 1983,* (1983), pp. 473–516.

[22] Jim Douglas and Todd Dupont. "Interior penalty procedures for elliptic and parabolic Galerkin methods". In: *Computing methods in applied sciences*. Springer, 1976, pp. 207–216.

[23] Jeff Erickson, Damrong Guoy, John M Sullivan, and Alper Üngör. "Building spacetime meshes over arbitrary spatial domains". In: *Engineering with Computers* 20.4 (2005), pp. 342–353.

[24] C. Farhat and P. Geuzaine. "Design and analysis of robust ALE time-integrators for the solution of unsteady flow problems on moving grids". In: *Comput. Methods Appl. Mech. Engrg.* 193.39-41 (2004), pp. 4073–4095. ISSN: 0045-7825.

[25] David A Field. "Laplacian smoothing and Delaunay triangulations". In: *Communications in applied numerical methods* 4.6 (1988), pp. 709–712.

[26] David A. Field. "Qualitative measures for initial meshes". In: *Internat. J. Numer. Methods Engrg.* 47 (2000), pp. 887–906.

[27] Bradley Froehle and Per-Olof Persson. "A high-order discontinuous Galerkin method for fluid–structure interaction with efficient implicit–explicit time stepping". In: *Journal of Computational Physics* 272 (2014), pp. 455–470.

[28] R Glowinski et al. "A fictitious domain approach to the direct numerical simulation of incompressible viscous flow past moving rigid bodies: application to particulate flow". In: *Journal of Computational Physics* 169.2 (2001), pp. 363–426.

[29] A. Guardone, D. Isola, and G. Quaranta. "Arbitrary Lagrangian Eulerian formulation for two-dimensional flows using dynamic meshes with edge swapping". In: *J. Comput. Phys.* 230.20 (2011), pp. 7706–7722. ISSN: 0021-9991.

[30] Jeffrey Hellrung, Luming Wang, Eftychios Sifakis, and Joseph Teran. "A second order virtual node method for elliptic problems with interfaces and irregular domains in three dimensions". In: *Journal of Computational Physics* 231.4 (2012), pp. 2015–2048.

[31] J. S. Hesthaven and T. Warburton. *Nodal discontinuous Galerkin methods*. Vol. 54. Texts in Applied Mathematics. Algorithms, analysis, and applications. New York: Springer, 2008, pp. xiv+500. ISBN: 978-0-387-72065-4.

[32] CW Hirt, Anthony A Amsden, and JL Cook. "An arbitrary Lagrangian-Eulerian computing method for all flow speeds". In: *Journal of Computational Physics* 14.3 (1974), pp. 227–253.

[33] Thomas JR Hughes and Gregory M Hulbert. "Space-time finite element methods for elastodynamics: formulations and error estimates". In: *Computer methods in applied mechanics and engineering* 66.3 (1988), pp. 339–363.

[34] Thomas JR Hughes, Wing Kam Liu, and Thomas K Zimmermann. "Lagrangian-Eulerian finite element formulation for incompressible viscous flows". In: *Computer methods in applied mechanics and engineering* 29.3 (1981), pp. 329–349.

[35] Gregory M Hulbert and Thomas JR Hughes. "Space-time finite element methods for second-order hyperbolic equations". In: *Computer Methods in Applied Mechanics and Engineering* 84.3 (1990), pp. 327–348.

[36] D. Isola and A. Guardone. "Simulation of flows with strong shocks with an adaptive conservative scheme". In: *J. Comput. Appl. Math.* 236.18 (2012), pp. 4660–4670. ISSN: 0377-0427.

[37] Claes Johnson. "Discontinuous Galerkin finite element methods for second order hyperbolic problems". In: *Computer Methods in Applied Mechanics and Engineering* 107.1 (1993), pp. 117–129.

[38] Samuel Kanner. "Design, Analysis and Hybrid Testing of Innovative Platforms for Floating Vertical-Axis Wind Turbines". PhD thesis. UNIVERSITY OF CALIFORNIA, BERKELEY, 2015.

[39] Dokyun Kim and Haecheon Choi. "Immersed boundary method for flow around an arbitrarily moving body". In: *Journal of Computational Physics* 212.2 (2006), pp. 662–680.

[40] C. M. Klaij, J. J. W. van der Vegt, and H. van der Ven. "Space-time discontinuous Galerkin method for the compressible Navier-Stokes equations". In: *J. Comput. Phys.* 217.2 (2006), pp. 589–611. ISSN: 0021-9991.

[41] Christiaan M Klaij, Jaap JW van der Vegt, and Harmen van der Ven. "Pseudo-time stepping methods for space–time discontinuous Galerkin discretizations of the compressible Navier–Stokes equations". In: *Journal of Computational Physics* 219.2 (2006), pp. 622–643.

[42] Christiaan M Klaij, Marc H van Raalte, Harmen van der Ven, and Jaap JW van der Vegt. "h-Multigrid for space-time discontinuous Galerkin discretizations of the compressible Navier–Stokes equations". In: *Journal of Computational Physics* 227.2 (2007), pp. 1024–1045.

[43] Sudeep K. Lahiri, Javier Bonet, and Jaume Peraire. "A variationally consistent mesh adaptation method for triangular elements in explicit Lagrangian dynamics". In: *Internat. J. Numer. Methods Engrg.* 82.9 (2010), pp. 1073–1113. ISSN: 0029-5981.

[44] Randall J LeVeque. *Finite volume methods for hyperbolic problems*. Vol. 31. Cambridge university press, 2002.

[45] Randall J LeVeque and Randall J Le Veque. *Numerical methods for conservation laws*. Vol. 132. Springer, 1992.

[46] I. Lomtev, R. M. Kirby, and G. E. Karniadakis. "A discontinuous Galerkin ALE method for compressible viscous flows in moving domains". In: *J. Comput. Phys.* 155.1 (1999), pp. 128–159. ISSN: 0021-9991.

[47] Robert B Lowrie, Philip L Roe, and Bram Van Leer. "Space-time methods for hyperbolic conservation laws". In: *Barriers and Challenges in Computational Fluid Dynamics*. Springer, 1998, pp. 79–98.

[48] Karthik Mani and Dimitri Mavriplis. "Efficient Solutions of the Euler Equations in a Time-Adaptive Space-Time Framework". In: *49th AIAA Aerospace Sciences Meeting and Exhibit*. AIAA-2011-774. 2011.

[49] Arif Masud and Thomas JR Hughes. "A space-time Galerkin/least-squares finite element formulation of the Navier-Stokes equations for moving domain problems". In: *Computer Methods in Applied Mechanics and Engineering* 146.1 (1997), pp. 91–126.

[50] Dimitri J Mavriplis and Cristian R Nastase. "On the geometric conservation law for high-order discontinuous Galerkin discretizations on dynamically deforming meshes". In: *Journal of Computational Physics* 230.11 (2011), pp. 4285–4300.

[51] Cesar A Acosta Minoli and David A Kopriva. "Discontinuous Galerkin spectral element approximations on moving meshes". In: *Journal of Computational Physics* 230.5 (2011), pp. 1876–1902.

[52] Vinh-Tan Nguyen, Jaime Peraire, Boo Cheong Khoo, and Per-Olof Persson. "A discontinuous Galerkin front tracking method for two-phase flows with surface tension". In: *Computers & Fluids* 39.1 (2010), pp. 1–14.

[53] Géraldine Olivier and Frédéric Alauzet. "A new changing-topology ALE scheme for moving mesh unsteady simulations". In: *49th AIAA Aerospace Sciences Meeting, AIAA Paper*. Vol. 474. 2011, pp. 252–271.

[54] J. Peraire and P.-O. Persson. "The compact discontinuous Galerkin (CDG) method for elliptic problems". In: *SIAM J. Sci. Comput.* 30.4 (2008), pp. 1806–1824. ISSN: 1064-8275.

[55] J. Peraire and Per-Olof Persson. "Adaptive High-Order Methods in Computational Fluid Dynamics". In: vol. 2. Advances in CFD. World Scientific Publishing Co., 2011. Chap. 5 – High-Order Discontinuous Galerkin Methods for CFD.

[56] P.-O. Persson. "Scalable Parallel Newton-Krylov Solvers for Discontinuous Galerkin Discretizations". In: *47th AIAA Aerospace Sciences Meeting and Exhibit, Orlando, Florida*. AIAA-2009-606. 2009.

[57] P.-O. Persson, J. Bonet, and J. Peraire. "Discontinuous Galerkin solution of the Navier-Stokes equations on deformable domains". In: *Comput. Methods Appl. Mech. Engrg.* 198 (2009), pp. 1585–1595.

[58] P.-O. Persson and J. Peraire. "Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier-Stokes equations". In: *SIAM J. Sci. Comput.* 30.6 (2008), pp. 2709–2733.

[59] P.-O. Persson, J. Peraire, et al. *The 3DG Project*. `http://persson.berkeley.edu`.

[60] P.-O. Persson and G. Strang. "A Simple Mesh Generator in MATLAB". In: *SIAM Review* 46 (2004).

[61] Per-Olof Persson. *Discontinuous Galerkin Methods for Fluid Flows and Implicit Mesh Generation.* Slides for Stanford FPC Fluid Mechanics Seminar. 2006.

[62] Per-Olof Persson. "Mesh Generation for Implicit Geometries". PhD thesis. M.I.T., Feb. 2005.

[63] Charles S Peskin. "The immersed boundary method". In: *Acta numerica* 11.0 (2002), pp. 479–517.

[64] G. Quaranta, D. Isola, and A. Guardone. "Numerical Simulation of the Opening of Aerodynamic Control Surfaces with Two-Dimensional Unstructured Adaptive Meshes". In: *5th European Conference on Computational Fluid Dynamics - ECCOMAS CFD 2010*. Vol. 236. 2010.

[65] Thomas C. S. Rendall, Christian B. Allen, and Edward D. C. Power. "Conservative unsteady aerodynamic simulation of arbitrary boundary motion using structured and unstructured meshes in time". In: *Internat. J. Numer. Methods Fluids* 70.12 (2012), pp. 1518–1542. ISSN: 0271-2091.

[66] Sander Rhebergen and Bernardo Cockburn. "A space-time hybridizable discontinuous Galerkin method for incompressible flows on deforming domains". In: *J. Comput. Phys.* 231.11 (2012), pp. 4185–4204. ISSN: 0021-9991.

[67] Sander Rhebergen, Bernardo Cockburn, and Jaap J. W. van der Vegt. "A space-time discontinuous Galerkin method for the incompressible Navier-Stokes equations". In: *J. Comput. Phys.* 233 (2013), pp. 339–358. ISSN: 0021-9991.

[68] P. L. Roe. "Approximate Riemann solvers, parameter vectors, and difference schemes". In: *J. Comput. Phys.* 43.2 (1981), pp. 357–372. ISSN: 0021-9991.

[69] PH Saksono, WG Dettmer, and D Perić. "An adaptive remeshing strategy for flows with moving boundaries and fluid–structure interaction". In: *International Journal for Numerical Methods in Engineering* 71.9 (2007), pp. 1009–1050.

[70] Jonathan Richard Shewchuk. "Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator". In: *Applied computational geometry towards geometric engineering*. Springer, 1996, pp. 203–222.

[71] Warnerius Egbert Hendrikus Sollie, O Bokhove, and JJW Van der Vegt. "Space–time discontinuous Galerkin finite element method for two-fluid flows". In: *Journal of Computational Physics* 230.3 (2011), pp. 789–817.

[72] K Stein, T Tezduyar, and R Benney. "Mesh moving techniques for fluid-structure interactions with large displacements". In: *Journal of Applied Mechanics* 70.1 (2003), pp. 58–63.

[73] James H Strickland, BT Webster, and T Nguyen. "A vortex model of the Darrieus turbine: an analytical and experimental study". In: *Journal of Fluids Engineering* 101.4 (1979), pp. 500–505.

[74] Shripad Thite. "Adaptive spacetime meshing for discontinuous Galerkin methods". In: *Computational Geometry* 42.1 (2009), pp. 20–44.

[75] PD Thomas and CK Lombard. "Geometric conservation law and its application to flow computations on moving grids". In: *AIAA journal* 17.10 (1979), pp. 1030–1037.

[76] Alper Üngör and Alla Sheffer. "Pitching tents in space-time: mesh generation for discontinuous Galerkin method". In: *Internat. J. Found. Comput. Sci.* 13.2 (2002). Volume and surface triangulations, pp. 201–221. ISSN: 0129-0541.

[77] Alper Üngör, Alla Sheffer, Robert B. Haber, and Shang-Hua Teng. "Layer based solutions for constrained space-time meshing". In: *Appl. Numer. Math.* 46.3-4 (2003). Applied numerical computing: grid generation and solution methods for advanced simulations, pp. 425–443. ISSN: 0168-9274.

[78] J. J. W. van der Vegt and J. J. Sudirham. "A space-time discontinuous Galerkin method for the time-dependent Oseen equations". In: *Appl. Numer. Math.* 58.12 (2008), pp. 1892–1917.

[79] J. J. W. van der Vegt and H. van der Ven. "Space-time discontinuous Galerkin finite element method with dynamic grid motion for inviscid compressible flows. I. General formulation". In: *J. Comput. Phys.* 182.2 (2002), pp. 546–585. ISSN: 0021-9991.

[80] H Van der Ven and JJW Van der Vegt. "Space–time discontinuous Galerkin finite element method with dynamic grid motion for inviscid compressible flows: II. Efficient flux quadrature". In: *Computer methods in applied mechanics and engineering* 191.41 (2002), pp. 4747–4780.

[81] Wolfgang A Wall et al. "Large deformation fluid-structure interaction–advances in ALE methods and new fixed grid approaches". In: *Fluid-structure interaction*. Springer, 2006, pp. 195–232.

[82] Decheng Wan and Stefan Turek. "Fictitious boundary and moving mesh methods for the numerical simulation of rigid particulate flows". In: *Journal of Computational Physics* 222.1 (2007), pp. 28–56.

[83] Hongwu Wang and Ted Belytschko. "Fluid–structure interaction by the discontinuous-Galerkin method for large deformations". In: *International Journal for Numerical Methods in Engineering* 77.1 (2009), pp. 30–49.

[84] Luming Wang and Per-Olof Persson. "A Discontinuous Galerkin Method for the Navier-Stokes Equations on Deforming Domains using Unstructured Moving Space-Time Meshes". In: *21st AIAA Computational Fluid Dynamics Conference, San Diego, California*. AIAA-2013-2833. 2013.

[85] Luming Wang and Per-Olof Persson. "A High-Order Discontinuous Galerkin Method with Unstructured Space-Time Meshes for Domains with Large Deformations". Under Review.

[86] Luming Wang and Per-Olof Persson. "High-order Discontinuous Galerkin Simulations on Moving Domains using an ALE Formulation and Local Remeshing with Projections". In: *53rd AIAA Aerospace Sciences Meeting, Orlando, Florida*. AIAA-2015-0820. 2015.

[87] Takahiro Yamada and Fumio Kikuchi. "An arbitrary Lagrangian-Eulerian finite element method for incompressible hyperelasticity". In: *Computer Methods in Applied Mechanics and Engineering* 102.2 (1993), pp. 149–177.

[88] Dehong Zeng and C Ross Ethier. "A semi-torsional spring analogy model for updating unstructured meshes in 3D moving domains". In: *Finite Elements in Analysis and Design* 41.11 (2005), pp. 1118–1139.