

RICE UNIVERSITY

GPU-accelerated discontinuous Galerkin methods on hybrid
meshes: applications in seismic imaging

by

Zheng Wang

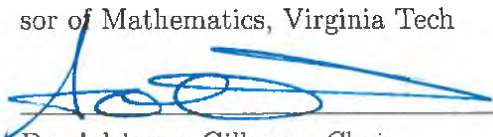
A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

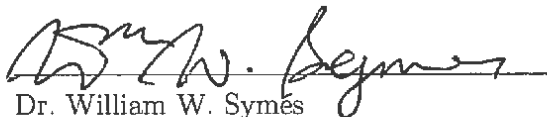
APPROVED, THESIS COMMITTEE:



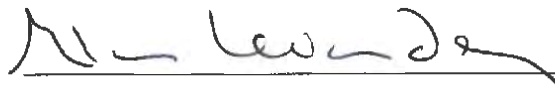
Dr. Timothy Warburton, *Director*
John K. Costain Faculty Chair and Profes-
sor of Mathematics, Virginia Tech



Dr. Adrianna Gillman, *Chair*
Assistant Professor of Computational and
Applied Mathematics, Rice University



Dr. William W. Symes
Noah Harding Professor of Computational
and Applied Mathematics, Rice University



Dr. Alan Levander
Carey Croneis Professor of Earth Science,
Rice University

HOUSTON, TEXAS
APRIL 2017

ABSTRACT

GPU-accelerated discontinuous Galerkin methods on hybrid meshes: applications in seismic imaging

by

Zheng Wang

Seismic imaging is a geophysical technique assisting in the understanding of subsurface structure on a regional and global scale. With the development of computer technology, computationally intensive seismic algorithms have begun to gain attention in both academia and industry. These algorithms typically produce high-quality subsurface images or models, but require intensive computations for solving wave equations.

Achieving high-fidelity wave simulations is challenging: first, numerical wave solutions may suffer from dispersion and dissipation errors in long-distance propagations; second, the efficiency of wave simulators is crucial for many seismic applications. High-order methods have advantages of decreasing numerical errors efficiently and hence are ideal for wave modelings in seismic problems.

Various high order wave solvers have been studied for seismic imaging. One of the most popular solvers is the finite difference time domain (FDTD) methods. The strengths of finite difference methods are the computational efficiency and ease of implementation, but the drawback of FDTD is the lack of geometric flexibility. It has been shown that standard finite difference methods suffer from first order numerical errors at sharp

media interfaces.

In contrast to finite difference methods, discontinuous Galerkin (DG) methods, a class of high-order numerical methods built on unstructured meshes, enjoy geometric flexibility and smaller interface errors. Additionally, DG methods are highly parallelizable and have explicit semi-discrete form, which makes DG suitable for large-scale wave simulations. In this dissertation, the discontinuous Galerkin methods on hybrid meshes are developed and applied to two seismic algorithms—reverse time migration (RTM) and full waveform inversion (FWI).

This thesis describes in depth the steps taken to develop a forward DG solver for the framework that efficiently exploits the element specific structure of hexahedral, tetrahedral, prismatic and pyramidal elements. In particular, we describe how to exploit the tensor-product property of hexahedral elements, and propose the use of hex-dominant meshes to speed up the computation.

The computational efficiency is further realized through a combination of graphics processing unit (GPU) acceleration and multi-rate time stepping. As DG methods are highly parallelizable, we build the DG solver on multiple GPUs with element-specific kernels. Implementation details of memory loading, workload assignment and latency hiding are discussed in the thesis. In addition, we employ a multi-rate time stepping scheme which allows different elements to take different time steps.

This thesis applies DG schemes to RTM and FWI to highlight the strengths of the DG methods. For DG-RTM, we adopt the boundary value saving strategy to avoid data movement on GPUs and utilize the memory load in the temporal updating procedure to produce images of higher

qualities without a significant extra cost. For DG-FWI, a derivation of the DG-specific adjoint-state method is presented for the fully discretized DG system. Finally, sharp media interfaces are inverted by specifying perturbations of element faces, edges and vertices.

Keyword: discontinuous Galerkin methods, high performance computing, reverse time migration, full waveform inversion

ACKNOWLEDGEMENTS

I would like first to thank my advisor — Dr. Tim Warburton. He leads me to the right direction of research, provides me a lot of opportunities and chooses the suitable projects for me. Research is a difficult way to go, but the right project is the half way to success. I have learned a lot in my project involving PDEs, GPUs and seismic imaging. The knowledge I obtained in this project also helps me find the internships and the final job.

I would also like to thank the committee members — Dr. Adrianna Gillman, Dr. William Symes and Dr. Alan Levander. They made useful comments and provided necessary help in my project. In addition, they taught wonderful classes which are also helpful for me.

I thank the former postdocs in our group — Dr. Jesse Chan and Dr. Axel Modave. They worked with me together in this project. Dr. Chan helped me with the DG methods and hybrid meshes, and Dr. Modave provided me the framework for coding and helped me understand the details of the code.

I want to thank my Rice friends, which includes but not limited to Yingpei Wang, Xin Yang, Jun Tan, Yangyang Xu, Xiaodi Deng, Jie Hou, Chen Liu, Arturo Vargas, Rajesh Gandham and David Medina. They helped me not only in the research but also in the daily life. I would like to give my special thanks to Yingpei and Xin who helped me a lot when I first came to the U.S.

Finally, I want to thank all my friends and relatives. Thanks for their supports along the way.

Contents

Abstract	ii
Acknowledgements	v
1 Introduction	1
1.1 Seismic imaging	2
1.2 Discontinuous Galerkin methods	8
1.3 GPU programming	12
1.4 Contributions	15
1.5 Outline	16
2 Discontinuous Galerkin method on hybrid meshes	17
2.1 Notational conventions	17
2.2 General formulation	19
2.3 Element types	24
2.4 Kernels and algorithms	34
2.5 Multi-rate time stepping	42
2.6 Working on clusters	49
2.7 Brief on data structure	53
2.8 Numerical convergence	56
2.9 Performance study	62
2.10 Summary	67
3 Reverse time migration	68
3.1 Preliminaries	70

3.2	Imaging condition	77
3.3	Backward phase of the source wavefield	80
3.4	Evaluation of the imaging condition	83
3.5	Numerical results of synthetic surveys	85
3.6	Summary	95
4	Full waveform inversion	97
4.1	Formulation	99
4.2	Continuous adjoint-state method for FWI	101
4.3	Discrete adjoint-state method for DG-FWI	108
4.4	Numerical results of conventional FWI	121
4.5	Inverting sharp interfaces	130
4.6	Summary	146
5	Summary and future work	148
	Bibliography	153

List of Figures

1.1	GPU 2D Grid	13
1.2	GPU Memory	13
1.3	OCCA API	14
2.1	2D illustration of plus and minus superscript	19
2.2	Hexahedron reference element	25
2.3	SEM and GL nodes	27
2.4	Tetrahedron reference element	29
2.5	Prism reference element	30
2.6	Pyramid reference element	33
2.7	Multi-rate time stepping: element grouping	46
2.8	Multi-rate time stepping: timeline	47
2.9	Multi-rate time stepping: updating procedure	48
2.10	Illustration of 3-level parallelism	50
2.11	MPI partition: 2D illustration	51
2.12	Asymmetric MPI communication	52
2.13	Mesh connectivity illustration	55
2.14	Patterns of mesh sequences	58
2.15	Convergence plots of individual element types	60
2.16	Hybrid mesh and convergence	61
2.17	Relative time cost of different element types	66
3.1	PML: 2D illustration	72
3.2	Source injection: 2D illustration	77

3.3	RTM Interpretation	80
3.4	Example of Ricker wavelet	86
3.5	Data muting in RTM configuration	86
3.6	Single layer example: velocity model	87
3.7	Single layer example: sources and receivers	88
3.8	Single layer example: results	89
3.9	Multi-layer example: velocity model	90
3.10	Multi-layer example: sources and receivers	90
3.11	Multi-layer example: results	91
3.12	Marmousi example: velocity model (3D view)	92
3.13	Marmousi example: velocity model (2D view)	93
3.14	Marmousi example: sources and receivers	93
3.15	Marmousi example: results (linear color scale)	94
3.16	Marmousi example: results (log color scale)	94
4.1	Adjoint test for the forward operator	117
4.2	Gaussian inclusion example: true velocity model.	122
4.3	Gaussian inclusion example: initial velocity model.	122
4.4	Gaussian inclusion example: sources and receivers (receivers are buried in the bottom)	122
4.5	Gaussian inclusion example: FWI results (receivers are buried in the bottom)	123
4.6	Gaussian inclusion example: history of misfit values receivers are buried in the bottom)	123
4.7	Gaussian inclusion example: sources and receivers (receivers are lo- cated near all boundaries)	124
4.9	Gaussian inclusion example: history of misfit values (receivers are lo- cated near all boundaries)	124
4.8	Gaussian inclusion example: FWI results (receivers are located near all boundaries)	125
4.10	Cubic inclusion example: true velocity model	126
4.11	Cubic inclusion example: initial velocity model	126
4.12	Cubic inclusion example: sources and receivers	126
4.13	Cubic inclusion example: FWI results	127
4.14	Cubic inclusion example: history of misfit values	127

4.15	Layer model example: true velocity model	128
4.16	Layer model example: initial velocity model	129
4.17	Layer model example: sources and receivers	129
4.18	Layer model example: FWI results	130
4.19	Layer model example: history of misfit values	130
4.20	Perturbation of the inclusion boundary	133
4.21	Cubic inclusion example: configuration	134
4.22	Cubic inclusion example: misfit function and its derivatives	135
4.23	Cubic inclusion example: mesh	136
4.24	Experiment of alternating update	137
4.25	Multi-surface example: configuration	138
4.26	Multi-surface example: mesh	139
4.27	Multi-surface example: true and initial models	139
4.28	Multi-surface example: solution	140
4.29	Illustration of perturbing vertex	141
4.30	Multi-vertex example: configuration	144
4.31	Multi-vertex example: mesh	144
4.32	Multi-vertex example: true and initial models	145
4.33	Multi-vertex example: solution	145

List of Tables

2.1	Convergence study of hexahedral elements	58
2.2	Convergence study of prismatic elements	59
2.3	Convergence study of pyramidal elements	59
2.4	Convergence study of tetrahedral elements	59
2.5	Configuration of hybrid meshes	62
2.6	Convergence study of hybrid meshes	62
2.7	Nvidia GTX 980 specification	63
2.8	GFLOPS and bandwidth of the kernels	64
2.9	Relative time cost of the kernels	65

Introduction

Seismology is the study of earthquakes and has contributed to the understanding of Earth interiors. The development of modern seismology can be traced back to the end of 19th century when researchers start to analyze the earthquake data using travel time information [1]. With the development of the oil industry, seismology, especially seismic imaging, began to play a role in the oil and gas exploration. Many seismic imaging algorithms have been developed in the last several decades, and these algorithms utilize the reflection of sound waves to reveal the underground structures. In a seismic survey, sound-wave sources are generated by airguns or explosives, and at the same time, seismic receivers are placed near the ground surface or sea surface to collect the wave information. Seismologists use the recorded observation of sound waves to deduce subsurface structures, which are used by geophysical interpreters to predict the location of crude oil and natural gas.

In the last several decades, the exponential growth in computational capability makes it feasible for computationally intensive algorithms to be employed in seismic imaging. For instance, some seismic imaging algorithms such as reverse time migration (RTM) and full waveform inversion (FWI) require the wave equations to be solved on large surveyed volumes and are computationally expensive.

The wave equations solved in seismic imaging are a class of hyperbolic partial differential equations (PDEs) describing the physics of wave propagation. Due to the difficulty of obtaining analytic solutions to wave equations on general domains, researchers and engineers typically solve wave equations with numerical approximations. A large variety of numerical wave solvers, each of which has specific features, have been developed. In this thesis, I study the discontinuous Galerkin time domain (DGTD) methods as a candidate discretization of the wave equations.

Due to the requirement of solving wave equations multiple times in some seismic algorithms, the efficiency of the wave solver is crucial in seismic imaging. To address the efficiency issue, the experimented DGTD solver is built on hybrid meshes containing hexahedra, tetrahedra, prisms and pyramids to exploit the strengths of different element types. The efficiency of the DG solver is also studied on modern hardware accelerators to further reduce the computational time cost. In this work, the graphics processing units (GPUs) are employed as the main computational hardware.

This chapter provides an overview of two seismic imaging algorithms—reverse time migration and full waveform inversion, and reviews DGTD methods and GPU programming.

1.1 Seismic imaging

In a seismic survey, sound waves are typically generated by explosives, and receivers such as geophones then detect and record the motion of the Earth as seismic data. The task of seismic imaging is to utilize the seismic data to produce images of the subsurface.

Various seismic algorithms haven been proposed to image the subsurface using seismic data. The computational costs and image qualities vary from algorithm to algorithm. Typically, the cheap algorithms (e.g. Kirchhoff migration [2] and trav-

eltime tomography [3]) produce low-quality results, while the expensive algorithms (e.g. reverse time migration [4], least-square reverse time migration [5] and full waveform inversion [6]) may lead to high-quality images. In this section, we focus on the introduction to reverse time migration (RTM) and full waveform inversion (FWI), both of which are relatively expensive seismic algorithms.

Reverse time migration is one of the seismic imaging algorithms that require solving wave equations many times. Originated from Claerbout's reflector mapping [7, 2], RTM was early introduced in the 1980s [4, 8], and has been a topic of research for decades [9]. At the first step, the wave equations must be efficiently solved in RTM.

In this thesis, I focus on the acoustic wave equation (one class of wave equations) in time domain, and its first order form is given by

$$\frac{\partial p(\mathbf{x}, t)}{\partial t} + \rho c^2 \nabla \cdot \mathbf{v}(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \mathbb{R}^d, t \in \mathbb{R}^+ \quad (1.1a)$$

$$\frac{\partial \mathbf{v}(\mathbf{x}, t)}{\partial t} + \frac{1}{\rho} \nabla p(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \mathbb{R}^d, t \in \mathbb{R}^+ \quad (1.1b)$$

where $p(\mathbf{x}, t)$ is the acoustic pressure, $\mathbf{v}(\mathbf{x}, t)$ is the particle velocity, $\rho(\mathbf{x})$ is the media density, $c(\mathbf{x})$ is the phase velocity, and $d = 1, 2, 3$ is the dimension. More general cases including elastic wave equations and frequency domain are referred to [6, 10].

A large variety of numerical methods are devoted to solving the wave equations. Some popular methods are listed below.

- Finite difference methods use Taylor expansion to approximate the derivatives of a function at a certain point. For instance, a spatial first order derivative may be approximated as

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h},$$

which is known as a central finite difference approximation [11]. Although weak

in dealing with complicated geometry [12], the finite difference method has a lot of applications in seismic imaging due to its simplicity and efficiency. Early work of finite difference in wave propagation can be traced back to 1970s [13, 14, 15], and it continues to be a popular tool in recent seismic research [16].

- Pseudo-spectral methods employ a set of basis functions to globally approximate PDE solutions. The pseudo-spectral method resembles spectral methods [17], but it takes advantage of fast Fourier transformation (FFT) which improves the computational efficiency [10]. The pseudo-spectral method is not easy to be parallelized due to its global approximation. Readers can refer to [18, 19] for a detailed introduction to pseudo-spectral methods.
- Finite element methods partition the computational domain into many geometric elements, and the solution is approximated in each element by a linear combination of basis functions [20, 21]. The use of unstructured mesh gives finite element methods geometric flexibility. However, the mass matrix of finite element methods must be inverted in an explicit time stepping scheme, and inverting the mass matrix is expensive. Techniques such as mass lumping may be introduced to avoid inverting the mass matrix, but it may also harm the accuracy of the solution [22]. An alternative to the standard finite element method is the spectral element method [23, 24, 25], which resembles finite element methods but usually employs Lagrange polynomials and Gauss-Legendre-Lobatto quadrature on rectangular and hexahedral elements. By restricting elements types, the spectral element methods sacrifice some mesh flexibility to achieve computational efficiency.
- Discontinuous Galerkin time domain methods, which will be discussed later in chapter 2, are suitable for seismic applications. In general, DG combines the

features of finite element methods and finite volume methods. Compared to finite difference methods, DG has smaller interface error [26] and is amenable to local adaptivity [27]. The studies of DG in seismology have been carried out by many researchers. As a wave propagator for seismic imaging, DG has been applied to acoustic, elastic, TTI and VTI problems [9, 28, 29, 30, 31].

In RTM, the wave equations are solved forward in time with source signals and backward in time by providing the recorded seismic data as the external forcing of the wave equation. The forward and backward solutions are then cross-correlated using

$$I(\mathbf{x}) = \int_0^T p_S(\mathbf{x}, t) p_R(\mathbf{x}, t) dt, \quad (1.2)$$

where p_S and p_R are the forward source field and backward receiver field respectively. Expression (1.2) is also known as the imaging condition. The output of RTM is this image $I(\mathbf{x})$ which reports the subsurface structures.

The original migration imaging principle was recast in the early 1980s [32, 33] as a local optimization problem which is known as the full waveform inversion. One version of the imaging condition, which resembles (1.2), can be used to build the gradient of the misfit function in the full waveform inversion.

Full waveform inversion is a procedure which utilizes the recorded data to recover the physical parameters of the subsurface. Based on the numerical solution to the wave equations, FWI iteratively improves the tomographic images using optimization techniques [10]. Different from RTM which only images the subsurface structures without realizing their physical properties, FWI does not only output the topology of subsurface structures, but also inverts the magnitude of physical parameters.

To begin the introduction to FWI, we generalize the representation of wave equations. Given the physical parameter $\mathbf{m}(\mathbf{x})$ (such as media density ρ and velocity c in equation (1.2)) with $\mathbf{x} \in \mathbf{R}^d$ ($d = 1, 2, 3$), we can solve the corresponding wave

equations to obtain the solution $\mathbf{u}(\mathbf{x}, t)$ (such as pressure p and particle velocity \mathbf{v} in (1.2)) with $\mathbf{x} \in \mathbb{R}^d$ and $t \in \mathbb{R}^+$. The physical parameter \mathbf{m} is called a model which gives physical properties of the subsurface; the solution \mathbf{u} is a wavefield which maps space and time to the wave attributes. The wave equation is viewed as an operator \mathcal{F} acting on \mathbf{m} to obtain \mathbf{u} . We then write $\mathcal{F}[\mathbf{m}] = \mathbf{u}$. During the wave propagation, signals are recorded at receiver locations, and these are denoted by data $\mathbf{d}(t)$. For example, if a receiver is placed at \mathbf{x}_r , we have $\mathbf{d}_r(t) = \mathbf{u}(\mathbf{x}_r, t)$. Typically, the data consists of the information from many receivers, and hence data $\mathbf{d}(t)$ is a collection of multiple data traces $\mathbf{d}_r(t)$.

In FWI, the model \mathbf{m} is the variable needs to be solved. The available information is the data \mathbf{d} produced by the true model \mathbf{m}_{true} and the corresponding source and receiver configurations. The task of FWI is to minimize the mismatch between the sampled synthetic wavefield $\mathcal{F}[\mathbf{m}]$ and the recorded data \mathbf{d} . To this end, a misfit function is given by

$$\begin{aligned} \mathcal{J}[\mathbf{m}] &= \frac{1}{2} \langle S\mathcal{F}[\mathbf{m}] - \mathbf{d}, S\mathcal{F}[\mathbf{m}] - \mathbf{d} \rangle \\ &= \frac{1}{2} \|S\mathcal{F}[\mathbf{m}] - \mathbf{d}\|_2^2, \end{aligned} \tag{1.3}$$

where S is a restriction operator that maps wavefields to traces at the receiver locations, $\|\cdot\|_2$ is the L^2 norm in space and time, and $\langle \cdot, \cdot \rangle$ is the corresponding inner product. In other words, we assume the equality $S\mathcal{F}[\mathbf{m}_{\text{true}}] = \mathbf{d}$ holds, and want to recover \mathbf{m}_{true} by solving the optimization problem (1.3).

The model \mathbf{m} is then obtained by minimizing the misfit function \mathcal{J} using an optimization algorithm. Widely used optimization methods in FWI are steepest-descent method, conjugate gradient method [34], Newton method, Gauss-Newton method [35] and Quasi-Newton method [36].

These optimization methods require derivatives of \mathcal{J} with respect to \mathbf{m} . The

derivatives of \mathcal{J} is costly to compute in a conventional entry-by-entry manner, but can be obtained through the adjoint-state method [10]. The adjoint-state method is a well-known method for PDE-constrained optimization. In the adjoint-state method, an adjoint variable, which is analogous to the Lagrangian multiplier, is introduced to guide the computation of the gradient of the misfit function. A review of the adjoint-state method in geophysical applications can be found in Plessix's 2006 paper [37]. For specific discrete problems, the adjoint-state method is dependent on the PDE discretizations, and hence the specific derivation of the adjoint-state method should be given for a certain class of problems. For example, Wilcox et al. [38] gave the discrete form of the adjoint-state method in the context of DG-based hyperbolic problems. In this thesis, I will also present a derivation of the adjoint-state method for DG methods in chapter 4.

The minimization problem in FWI is ill-posed, and the FWI problem is not guaranteed to have a unique solution [6]. An example to explain this is a 1D reflection model: assuming the velocity model has two different materials and there is one source to emit a signal and one receiver to collect the reflected wave, we can always tune the magnitude of the velocity model and the location of the reflector to produce the same data. In addition, the misfit function in the FWI formulation may contain many local minimums. As a local optimization scheme is typically used for FWI, the numerical solutions may be trapped in local minima. This is sometimes known as the cycle skipping issue [6]. Many techniques are proposed to formulate a better-posed problem. Examples are using a regularization term to guide the optimization paths [34, 39], using a different objective function [40], and applying frequency continuation technique [41]. Due to the ill-posedness of the FWI formulation, the initial model for FWI must be good enough to avoid the cycle skipping issue. Several algorithms can be used to provide an initial guess such as reflection tomography [42], first-arrival

traveltime tomography [3], and Laplace-domain inversion [43].

1.2 Discontinuous Galerkin methods

Both RTM and FWI require efficient solvers for wave equations. In this thesis, I explore the discontinuous Galerkin (DG) methods as potential wave solvers. As a class of numerical methods for solving partial differential equations, discontinuous Galerkin methods resemble finite element methods in formulation but allow discontinuity among elements. Introduced in 1973 by Reed and Hill [44], DG methods have been applied to a diverse range of fields such as porous media flows [45], viscoelastic flows [38], modeling of shallow water [46], electromagnetism [47], wave propagation [9], and many others [48, 45]. DG methods have advantages over the finite element, finite volume and finite difference in several aspects [48, 47]:

- DG is a mesh-based method, and hence the unstructured mesh can handle complicated geometries especially media interfaces. Symes and Vdovina [12] noticed that the standard finite difference method has first order interface error. When the media has sharp interfaces, the convergence rate of FD becomes only first order, thus destroying the high order convergence from the spatial discretization. In contrast, Wang [26] numerically proved that DG has high order interface errors when the mesh is aligned with the media interface.
- The mass matrices generated in DG formulation are block-diagonal, which saves the computational cost of solving a linear system when using an explicit time-stepping scheme.
- DG methods are particularly efficient with high order discretization, thus suitable for high-frequency wave simulations.

- DG methods are highly parallelizable, which makes DG well suited for large-scale simulations in seismic imaging [9].

As a mesh-based method, DG methods solve PDEs on complicated geometries partitioned by meshes. In particular, it is the triangle in 2 dimensional (2D) spaces and the tetrahedron in 3 dimensional (3D) spaces that can easily approximate complicated geometries. In this work, I employ the nodal discontinuous Galerkin methods developed by Hesthaven and Warburton [47] to formulate a discretized system on tetrahedral elements in 3D domains. The nodal DG technique has been applied to many physics problems [49, 46, 50, 51]. In particular, its application in acoustic/elastic waves, which is also the interest of this thesis, has been studied by many researchers including Wang [26], Modave [9], Wilcox [31] and Matar [52].

Besides the nodal DG approach, various DG formulations have been studied for wave problems. In 2003, Rivière and Wheeler [53] analyzed nonsymmetric interior penalty DG formulation for second order wave equations, and it was also extended to the symmetric interior penalty by Grote and others [54]. The dispersive and dissipative properties of this interior penalty DG formulation, together with the properties of the general DG schemes, were studied by Ainsworth, Monk and Muniz [55]. In 2006, Dumbser and Käser applied DG methods with orthogonal bases to elastic wave equations, where arbitrary high-order derivatives for flux calculation was used. In 2015, Chan and Warburton [56] proposed Bernstein-Bezier DG methods for wave problems. With the assumption of piece-wise constant physical parameters, the Bernstein-Bezier DG methods introduced sparsity in the element-wise operators and improved the overall performance on tetrahedral meshes. In 2016, Chan, Hewett and Warburton developed weight-adjusted discontinuous Galerkin methods to reduce computational cost for wave equations with smooth coefficients [57].

While tetrahedral elements benefit from their flexibility, hexahedral elements enjoy a low computational cost brought by their tensor product property [58]. The DG methods on hexahedral elements resemble spectral element methods (SEM) since their solutions are both approximated by high degree Lagrange interpolants. Simulations and analysis of SEM on wave propagation were studied in the 1990s [59, 60]. The DG methods on hexahedra inherit many advantages from SEM and provide a wider choice for basis functions by removing the continuity enforcement. However, the mesh of only hexahedral elements is not amenable to complicated geometries, so a combination of tetrahedral and hexahedral elements is a better choice regarding both flexibility and efficiency. In a hybrid mesh, tetrahedral elements can be utilized to approximate the complicated geometry, and hexahedral elements should fill the rest of the domain to improve the efficiency and save the computational time cost. Since a tetrahedron only has triangular faces while a hexahedron only has quadrangular faces, prisms and pyramids must be introduced to glue tetrahedra and hexahedra in a conforming mesh.

Early explorations of hybrid meshes using spectral/hp methods were carried out by Sherwin [61], Warburton [62, 63] and Kirby [64]. Their work introduced a local coordinate system, constructed orthogonal expansions, and studied applications mainly in computational fluid dynamics and magnetohydrodynamics. The construction of nodal elements on hybrid meshes (including polyhedra) was discussed by Gassner et al. [65]. While the approximation spaces for hexahedral, prismatic and tetrahedral elements from the early research have been widely accepted, the improvement of basis functions for the pyramid has continuously been a research topic. In 1992, Bedrosian [66] realized that polynomial basis functions suffer from the singularity issue in a vertex-mapped pyramid, and he thereby constructed low order basis for pyramids using rational expressions. After that, Bergot, Cohen and Duruffe introduced their high

order function basis for pyramids in [67, 68]. More recently, Chan and Warburton [69] derived orthogonal bases for pyramidal elements which enabled diagonal mass matrix in discontinuous Galerkin methods.

Due to the computational efficiency of hexahedral elements, hex-dominant meshes are well suited for large simulations, and hence various techniques were proposed to assist the computation on hybrid meshes. In order to minimize the storage for time-explicit DG methods, a low-storage curvilinear discontinuous Galerkin (LSC-DG) method for prisms was introduced by Warburton [70, 71]. By dividing the basis function by the square root of the Jacobian, the LSC-DG transforms the mass matrix into an identity and hence avoids storing the mass matrix. For pyramids, the basis functions can be constructed as orthogonal bases [69], which results in a diagonal mass matrix and saves computational storage and time.

To further improve the computational efficiency, a multi-rate time stepping technique has been adopted in my DG implementation. The multi-rate time stepping allows different elements to take different time step sizes, which relaxes the global CFL constraints into many local CFL constraints. The multi-rate time stepping technique has been studied by many researchers including Gear and Wells [72], Warburton [73], Gödel [74], Gandham [46] and Modave [9].

This thesis continues the effort to develop discontinuous Galerkin methods on hybrid meshes and utilizes computational techniques including multi-rate time stepping and multi-GPU implementation to accelerate the computation. In particular, we address the computational efficiency of hex-dominant meshes. The implementation of this thesis is extended from a simplified version of **RiDG** code [9], which is a GPU-accelerated C++ code for DG-RTM.

1.3 GPU programming

One of the research objectives is the ability to run large 3D simulations, which requires large operational capability. To improve time to solutions, I employ parallel computing with multiple hardware accelerators that can be central processing units (CPUs), graphics processing units (GPUs), or other hardware. Although my implementation runs on both CPUs and GPUs, in this work I focus on the GPU programming and describe the implementation details on GPUs. In the remaining of this thesis, I refer hardware accelerators to GPUs when there is no danger of confusion.

A graphics processing unit (GPU) is a multi-thread architecture, which is initially designed for graphical rendering but now also used in scientific computation. The applications of GPU have been involved in various fields such as reservoir simulation [75], geoscience [9, 76], computational fluid dynamics [77], machine learning [78], and finance [79].

A GPU consists of a series of streaming multiprocessors (SMs), and each SM contains registers, caches, warp schedulers and execution cores [80]. It is the SMs that performs the actual computations, and they are mapped to virtual grids, blocks, and threads at runtime. As illustrated in figure 1.1, the entire computational task is built on the GPU grid. The grid is then divided into many blocks, and the blocks are divided into many threads. When programming on GPUs, programmers write code for the threads, and the instructions will be executed simultaneously on the threads. Each thread is mapped to a GPU core. The GPU cores are physically grouped into warps, and each block may correspond to one or several warps.

Threads can access different levels of memory based on their virtual positions. As depicted in figure 1.2, each thread has its private memory which is also known as registers. The threads in the same block can read and write the corresponding shared memory. All threads can access the global memory while the global memory acts as

a general buffer between the host (CPU) and the device (GPU).

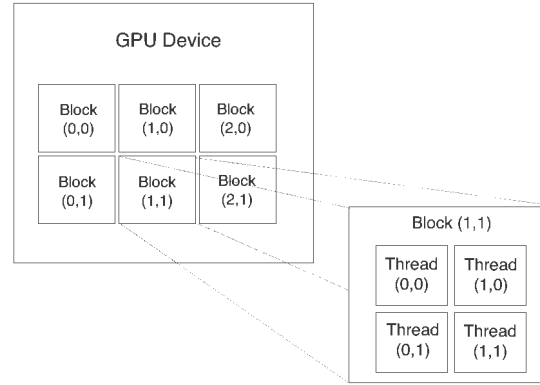


Figure 1.1: GPU Grid and Block: 2D Example. Threads are grouped into blocks, and blocks are arranged in a grid. Figures are regenerated from [81].

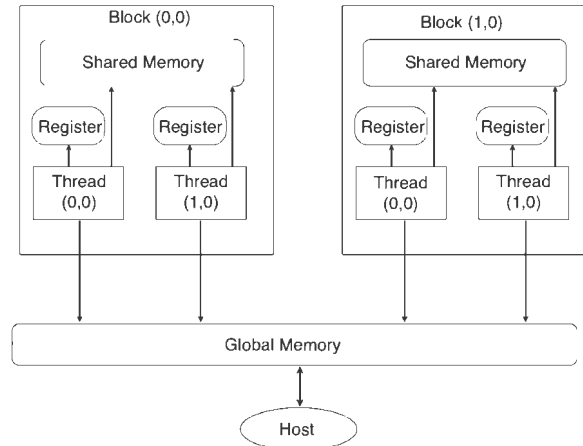


Figure 1.2: GPU Memories: 2D Example. A thread has its private memory, a block has its shared memory and every thread can access the global memory. Figures are regenerated from [81].

Several application programming interfaces (APIs), such as OpenCL and CUDA, are available for programmers to write GPU codes. OpenCL can be implemented on a variety of hardware provided by many vendors such as NVIDIA, AMD and Intel. The performance of OpenCL codes depends on the vendor implementation of OpenCL. Alternatively, CUDA may be efficient in parallelization on NVIDIA GPUs.

To take advantage of different programming approaches, I use OCCA, a unified

approach to multithreading languages, as the programming tool in the thesis study. In brief, OCCA is a syntax unification, and it provides a class of unified APIs for different programming modes. The OCCA keywords can be translated into an OpenMP, OpenCL or CUDA syntax at runtime (figure 1.3). Further details of OCCA language are referred to [82].

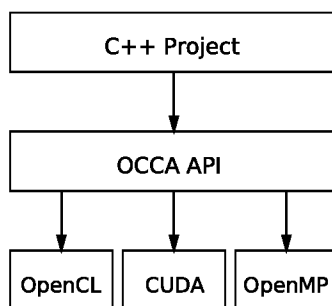


Figure 1.3: OCCA wraps different APIs. OCCA code is translated into an OpenMP, OpenCL or CUDA code at runtime.

With the rapid growth of the computational capacity of GPUs, researches of GPU-DG has been carried out in recent years. The GPU-DG has a variety of applications including electromagnetic fields [83], elastic waves [84], shallow water equations [46], reverse time migration [9] and many others [85, 86, 87]. Besides these applications, GPU-DG/FEM was also studied on different meshes: the DG method using tetrahedral mesh on a single GPU was implemented in 2009 by Klöckner, Warburton, Bridge and Hesthaven [49]. The spectral finite element method with hexahedral elements on GPUs was exploited in 2015 by Remacle, Gandham and Warburton [58]. Recent research on GPU-accelerated DG using hybrid meshes was also presented in 2015 by Chan, Wang, et al. [88].

Although GPUs are powerful computational devices, a single GPU is still limited in its computational capability. To this end, multiple GPUs can be connected to ful-

fill large computational tasks. The connected GPUs may communicate via message passing interface (MPI) which is a standard library specification for data movement among parallel devices. In recent years, the programming model of MPI+X is proposed for large-scale computations [89], where X stands for any parallelization on a shared memory system. As MPI+X is already the current approach for large-scale computations and considered to be a future trend for even larger computations, I use MPI+OCCA to enable multi-GPU simulations. The wave propagation modeling on GPU clusters has been studied by many researchers using various methods including finite difference methods [90], finite element methods [76, 91], and discontinuous Galerkin methods [9].

1.4 Contributions

This thesis introduces the DG method for acoustic waves, the details of GPU implementation, and its applications in RTM and FWI. The novelty of this thesis lies in the following aspects.

- A DG solver on hybrid meshes is developed. The solver is equipped with multi-rate time stepping and multi-GPU acceleration (MPI+OCCA). The efficiency of different element types is studied which suggest that a hex-dominant mesh is both efficient and flexible for large-scale simulations.
- The DG based RTM and FWI are developed. RTM images and FWI velocity models have been successfully produced by our software, which validates the correctness of our implementation.
- A derivation of the discrete adjoint-state method for DG-FWI using single rate Adams-Bashforth scheme is presented in chapter 4. The results of my derivation agree with Wilcox et al. [38], but may be more easily generalized.

- Sharp interfaces are demonstrated to be successfully inverted inside the DG-FWI framework. The impact of a certain interface perturbation is evaluated by integrating the imaging condition over element faces. Simple geometries such as polyhedral inclusions are inverted under certain assumptions. In addition, meshes are regenerated at FWI iterations to be aligned with the media interfaces.

1.5 Outline

The remaining chapters in this thesis are structured as follows.

- Chapter 2 is a detailed description of the discontinuous Galerkin methods for the acoustic wave equations. A formal introduction and discussion regarding the GPU implementation is presented. Numerical experiments with known analytic solutions are presented to validate the implementation. Performance studies are reported to analyze our implementation.
- Chapter 3 introduces the reverse time migration including the formulation and the implementation details. Test cases are given to validate the implementation.
- Chapter 4 discusses the application of DG in full waveform inversion. The adjoint-state method is firstly reviewed. Conventional FWI is then implemented and validated with test cases. Last but not the least, media interfaces are inverted by specifying a certain perturbation of the interfaces.
- Chapter 5 summarizes the thesis and proposes future works.

Discontinuous Galerkin method on hybrid meshes

Discontinuous Galerkin (DG) methods were initially proposed for the neutron transport equation [44], and have been applied to general hyperbolic equations [47]. The studies of DG methods on wave problems, hybrid meshes and GPUs have been discussed in chapter 1, and a more general review of DG methods can be found in [48]. In DG methods, the computational domain is partitioned into many elements, and the solution is typically represented as piecewise continuous polynomials. For linear PDEs, a linear system is then built from DG discretization and solved to obtain a numerical solution. This chapter introduces the formulation of the DG methods, the properties of different element types, implementation details, parallelization strategy, convergence study and performance study.

2.1 Notational conventions

As preliminaries, I discuss nomenclature:

- Scalars: regular letters are usually scalars in \mathbb{R} except some capitalized letters denote matrices. For instance, the wavefield pressure p , the time variable t and the phase velocity c are all scalars.

- Vectors: the vectors in \mathbb{R}^n are denoted by the bold-faced symbols. For example, \mathbf{x} usually refers to a coordinate vector in \mathbb{R}^3 , and the solution field vector of the acoustic waves is denoted by $\mathbf{u} = (p, v_1, v_2, v_3)$.
- L^2 inner product: round bracket denotes the L^2 inner product. It usually comes with a subscript indicating the integral domain. With this definition, the integral of u times v over the domain Ω is written as

$$(u, v)_\Omega = \int_\Omega uv \, d\mathbf{x}.$$

- Vector inner product: the discrete inner product of two vectors are denoted by dot product as

$$\mathbf{u} \cdot \mathbf{v} = \sum_i u_i v_i.$$

- Plus and minus superscript: in DG methods, we use a plus and minus superscript to denote the exterior and interior trace of a field. Since the DG solution overlaps at element faces, we use u^- to denote u inside the active element, and u^+ to denote u from the neighboring element (figure 2.1). Mathematically, we have

$$u^+ = \lim_{t \rightarrow 0^+} u(\mathbf{x} + t\mathbf{n}), \quad u^- = \lim_{t \rightarrow 0^-} u(\mathbf{x} + t\mathbf{n}),$$

where \mathbf{x} is on the element boundary, and \mathbf{n} is the outward facing unit normal to a face.

- Average and jump: double curly bracket and double square bracket are notations for average and jump respectively. The definition of average is consistent in both scalars and vectors, and it is defined by

$$\{\{u\}\} = \frac{u^- + u^+}{2}.$$

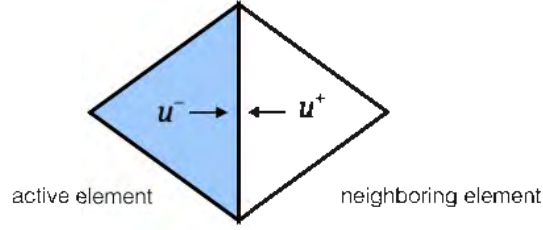


Figure 2.1: 2D illustration of plus and minus superscript: u^- denotes the traces on the active element while u^+ is the trace on the neighboring element.

The definition of normal jump is different for scalars and vectors. The jump of a scalar is a vector while the jump of a vector becomes a scalar. The mathematical form is given by

$$[[u]] = u^- \mathbf{n}^- + u^+ \mathbf{n}^+, \quad [[\mathbf{u}]] = \mathbf{u}^- \cdot \mathbf{n}^- + \mathbf{u}^+ \cdot \mathbf{n}^+,$$

where \mathbf{n}^- is the outward normal to an interior face, and \mathbf{n}^+ is the outward normal to an exterior face.

2.2 General formulation

This section studies the DG formulation. Here, we only consider the acoustic wave equation, but it may be extended to general hyperbolic equations. In the following, we will discuss the weak form and strong form formulation and derive the upwind fluxes for the acoustic wave equation.

2.2.1 Weak form and strong form

The DG scheme is similar to finite element method in formulation. To illustrate it, we apply the DG scheme to the 3D acoustic wave equation which is given by

$$\frac{\partial p(\mathbf{x}, t)}{\partial t} + \rho(\mathbf{x})c(\mathbf{x})^2 \nabla \cdot \mathbf{v}(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \Omega, \quad (2.1a)$$

$$\frac{\partial \mathbf{v}(\mathbf{x}, t)}{\partial t} + \frac{1}{\rho(\mathbf{x})} \nabla p(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \Omega, \quad (2.1b)$$

$$p(\mathbf{x}, 0) = p_0, \quad \mathbf{x} \in \Omega, \quad (2.1c)$$

$$\mathbf{v}(\mathbf{x}, 0) = \mathbf{v}_0, \quad \mathbf{x} \in \Omega, \quad (2.1d)$$

where $\Omega \subset \mathbb{R}^3$ is the domain of interest, \mathbf{x} is the spacial coordinate, t is the time, p and $\mathbf{v} = (v_1, v_2, v_3)$ are the pressure and velocity of the wavefield respectively, ρ is the media density and c is the phase velocity. When Ω is a bounded domain, equation (2.1) must be equipped with a proper boundary condition. For instance, a Dirichlet boundary condition may be given by

$$p(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \Gamma, \quad (2.2)$$

where $\Gamma = \partial\Omega$ is the boundary of the domain.

In the DG formulation, the global solution is typically approximated by piecewise polynomials. To begin with, the computational domain Ω is partitioned into many elements as

$$\Omega = \cup_{k=1}^K D^k. \quad (2.3)$$

On each element D^k , a local approximation is defined by

$$u_h^k(\mathbf{x}, t) = \sum_{n=1}^{N_p} \hat{u}_n^k(t) \varphi_n^k(\mathbf{x}), \quad (2.4)$$

where $\hat{u}_n^k(t)$ are coefficients depending on time and $\varphi_n^k(\mathbf{x})$ are basis functions supported on D^k . The global numerical solution is defined by the direct sum of all the local approximations

$$u_h = \oplus_{k=1}^K u_h^k. \quad (2.5)$$

The unknowns are the \hat{u}_n^k in equation (2.4) which determine the DG solution.

Replacing p and \mathbf{v} with discrete approximation p_h and $\mathbf{v}_h = (v_{1,h}, v_{2,h}, v_{3,h})$ in equation (2.1), multiplying the equation with test functions q and \mathbf{w} , and integrating over an element D^k , we obtain

$$\int_{D^k} \frac{\partial p_h}{\partial t} q + \int_{D^k} \rho c^2 \nabla \cdot \mathbf{v}_h q = 0, \quad (2.6a)$$

$$\int_{D^k} \rho \frac{\partial v_{i,h}}{\partial t} w_i + \int_{D^k} \frac{\partial p_h}{\partial x_i} w_i = 0, \quad i = 1, 2, 3. \quad (2.6b)$$

Spacial integration of equation (2.6) yields,

$$\int_{D^k} \frac{\partial p_h}{\partial t} q - \int_{D^k} \rho c^2 \mathbf{v}_h \cdot \nabla q = - \int_{\partial D^k} \rho c^2 \mathbf{v}_h \cdot \mathbf{n} q, \quad (2.7a)$$

$$\int_{D^k} \rho \frac{\partial v_{i,h}}{\partial t} w_i - \int_{D^k} p_h \frac{\partial w_i}{\partial x_i} = - \int_{\partial D^k} p_h w_i n_i, \quad i = 1, 2, 3. \quad (2.7b)$$

where $\mathbf{n} = (n_1, n_2, n_3)$ is the outward normal of ∂D^k .

Since the definition of a DG solution in (2.4) and (2.5) are not uniquely defined on the boundary of an element, the right-hand sides of equation (2.7) are not well defined. Therefore, we replace p_h and \mathbf{v}_h in the face integral in (2.7) with p_h^* and \mathbf{v}_h^* which are known as numerical fluxes [47]. The fluxes p_h^* and \mathbf{v}_h^* depend on the traces $p_h^+, p_h^-, \mathbf{v}_h^+$ and \mathbf{v}_h^- . The derivation of the numerical fluxes is given in section 2.2.2.

Assuming the numerical fluxes are already given, we have

$$\int_{D^k} \frac{\partial p_h}{\partial t} q - \int_{D^k} \rho c^2 \mathbf{v}_h \cdot \nabla q = - \int_{\partial D^k} \rho c^2 \mathbf{v}_h^* \cdot \mathbf{n} q, \quad (2.8a)$$

$$\int_{D^k} \rho \frac{\partial v_{i,h}}{\partial t} w_i - \int_{D^k} p_h \frac{\partial w_i}{\partial x_i} = - \int_{\partial D^k} p_h^* w_i n_i, \quad i = 1, 2, 3. \quad (2.8b)$$

This formulation, obtained from integration by part once, is referred to as the weak form formulation. Alternatively, a strong form formulation can be derived by integrating by part again, which leads to

$$\int_{D^k} \frac{\partial p_h}{\partial t} q + \int_{D^k} \rho c^2 \nabla \cdot \mathbf{v}_h q = \int_{\partial D^k} \rho c^2 (\mathbf{v}_h - \mathbf{v}_h^*) \cdot \mathbf{n} q, \quad (2.9a)$$

$$\int_{D^k} \rho \frac{\partial v_{i,h}}{\partial t} w_i + \int_{D^k} \frac{\partial p_h}{\partial x_i} w_i = \int_{\partial D^k} (p_h - p_h^*) w_i n_i, \quad i = 1, 2, 3, \quad (2.9b)$$

where \mathbf{v}_h and p_h in the face integral on the right-hand side take values in the interior of element D^k .

Both (2.8) and (2.9) are proper DG formulations, and they are mathematically equivalent with the assumption that test functions are smooth. However, the weak form formulation require differentiability of the test functions (q, \mathbf{w}) but not the numerical solutions (p_h, \mathbf{v}_h) , while the strong form formulation does not require smoothness of the test functions (q, \mathbf{w}) . A combination of strong form and weak form formulations may be used for different purposes: Warburton [71] took half of each formulation to guarantee stability for inexact quadrature. Wilcox et al. [38] applied strong and weak formulations alternatively in the adjoint-state method to derive discretely exact gradient. In this study, I use different formulations for different types of element. For hexahedron and tetrahedron, I use the strong form formulation due to the ease of implementation. For prism and pyramid, I apply a skew-symmetric formulation, which is derived by taking half strong form (2.9b) and half weak form (2.8a), to guarantee

stability [88]. Details of the skew-symmetric formulation are given in section 2.3.3.

2.2.2 Numerical flux

The formulation is complete once we properly define the numerical fluxes in equation (2.8) and (2.9). To this end, we follow the procedure in section 2.4 of the nodal DG book [47], and give a brief mathematical derivation of the numerical fluxes.

Denoting $\mathbf{u} = (p, \mathbf{v}) = (p, v_1, v_2, v_3)$, we first write the acoustic wave equation in a standard form of hyperbolic equation

$$Q \frac{\partial \mathbf{u}}{\partial t} + A_1 \frac{\partial \mathbf{u}}{\partial x_1} + A_2 \frac{\partial \mathbf{u}}{\partial x_2} + A_3 \frac{\partial \mathbf{u}}{\partial x_3} = 0 \quad (2.10)$$

with

$$Q = \begin{pmatrix} \frac{1}{\rho c^2} & & & \\ & \rho & & \\ & & \rho & \\ & & & \rho \end{pmatrix}, A_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & & & \\ 0 & & & \\ 0 & & & \end{pmatrix}, A_2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & & & \\ 1 & & & \\ 0 & & & \end{pmatrix}, A_3 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & & & \\ 0 & & & \\ 1 & & & \end{pmatrix}.$$

Define operator $\Pi = n_1 A_1 + n_2 A_2 + n_3 A_3$, where $\mathbf{n} = (n_1, n_2, n_3)$ is the outward normal vector. Following the formula in [47], we have

$$\begin{aligned} c^- Q^- (\mathbf{u}^* - \mathbf{u}^-) + (\Pi \mathbf{u})^* - (\Pi \mathbf{u})^- &= 0, \\ -c^+ Q^+ (\mathbf{u}^* - \mathbf{u}^+) + (\Pi \mathbf{u})^* - (\Pi \mathbf{u})^+ &= 0. \end{aligned} \quad (2.11)$$

Manipulation yields

$$(c^+ Q^+ + c^- Q^-) (\Pi \mathbf{u})^* = c^+ Q^+ (\Pi \mathbf{u})^- + c^- Q^- (\Pi \mathbf{u})^+ + c^- c^+ Q^- Q^+ (\mathbf{u}^- - \mathbf{u}^+), \quad (2.12)$$

which is simplified as

$$\begin{aligned}
\mathbf{v}^* &= \frac{1}{\rho^+ c^+ + \rho^- c^-} ((\rho c)^- \mathbf{v}^- + (\rho c)^+ \mathbf{v}^+ + (p^- - p^+) \mathbf{n}) \\
&= \frac{2}{\rho^+ c^+ + \rho^- c^-} \left(\{\{\rho c \mathbf{v}\}\} + \frac{1}{2} \llbracket p \rrbracket \right), \\
p^* &= \frac{1}{c^+ \rho^+ + c^- \rho^-} (c^+ \rho^+ p^- + c^- \rho^- p^+ + c^+ c^- \rho^+ \rho^- \mathbf{n} \cdot (\mathbf{v}^- - \mathbf{v}^+)) \\
&= \frac{2c^+ c^- \rho^+ \rho^-}{c^+ \rho^+ + c^- \rho^-} \left(\{\{p/(c\rho)\}\} + \frac{1}{2} \llbracket \mathbf{v} \rrbracket \right).
\end{aligned} \tag{2.13}$$

This result is also known as upwind fluxes [47].

2.3 Element types

Section 2.2 gives a general DG formulation, which applies to all types of elements. However, the specific properties of different element types should be considered in order to exploit their computational strengths. The hexahedral element is cheap in computation due to its tensor product property [17]; the tetrahedral element can easily approximate complicated geometries, benefiting from the mature tetrahedral mesh generation techniques [92]; the prismatic and pyramidal elements have both triangular and quadrangular faces, and hence act as transitional elements in the computational domain [88]. For completeness, this section highlights the different properties of each element type.

2.3.1 Hexahedron

The reference hexahedron is mathematically represented as

$$hex_{ref} = \{(r, s, t) \mid -1 \leq r, s, t \leq 1\}. \tag{2.14}$$

A reference hexahedron is mapped to a straight-side hexahedron with vertices $\mathbf{v}^1 \sim \mathbf{v}^8$ by

$$\begin{aligned} \mathbf{x} = & \frac{(1-r)}{2} \frac{(1-s)}{2} \frac{(1-t)}{2} \mathbf{v}^1 + \frac{(1+r)}{2} \frac{(1-s)}{2} \frac{(1-t)}{2} \mathbf{v}^2 \\ & + \frac{(1+r)}{2} \frac{(1+s)}{2} \frac{(1-t)}{2} \mathbf{v}^3 + \frac{(1-r)}{2} \frac{(1+s)}{2} \frac{(1-t)}{2} \mathbf{v}^4 \\ & + \frac{(1-r)}{2} \frac{(1-s)}{2} \frac{(1+t)}{2} \mathbf{v}^5 + \frac{(1+r)}{2} \frac{(1-s)}{2} \frac{(1+t)}{2} \mathbf{v}^6 \\ & + \frac{(1+r)}{2} \frac{(1+s)}{2} \frac{(1+t)}{2} \mathbf{v}^7 + \frac{(1-r)}{2} \frac{(1+s)}{2} \frac{(1+t)}{2} \mathbf{v}^8. \end{aligned} \quad (2.15)$$

Denoting a line segment in \mathbb{R} as

$$line_{ref} = \{r \mid -1 \leq r \leq 1\}, \quad (2.16)$$

we can write the hexahedron as a tensor product of three line segments

$$hex_{ref} = line_{ref} \otimes line_{ref} \otimes line_{ref}. \quad (2.17)$$

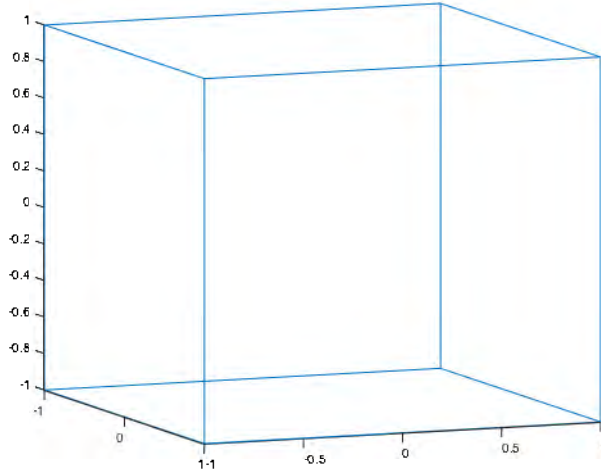


Figure 2.2: The hexahedron reference element.

This geometric tensor product property characterized in (2.17) leads to an alge-

braic tensor product property for basis functions on hexahedron, which can be utilized for fast computation.

To take advantage of the tensor product property, the basis functions on the hexahedral reference element are defined by a product of 1D basis functions,

$$\hat{\varphi}_{ijk}(r, s, t) = \hat{l}_i(r)\hat{l}_j(s)\hat{l}_k(t), \quad (2.18)$$

with $\hat{l}_n(x)$ being the Lagrange polynomials given by

$$\hat{l}_n(x) = \prod_{m \neq n} \frac{x - x_m}{x_n - x_m}, \quad (2.19)$$

where x_i can be either Gauss-Legendre (GL) nodes or Gauss-Legendre-Lobatto (GLL) nodes. The basis functions constructed on the GL nodes are orthogonal on affine mapped elements, which results in a diagonal mass matrix on a structured grid, but we may lose this orthogonality on arbitrary elements. In contrast, the GLL nodes, known as the basis for spectral element methods [93, 17], contains the endpoints -1 and 1 , and the mass matrix generated by the GLL nodes is not strictly diagonal on reference elements. More specifically, the quadrature rule based on GL nodes is exactly accurate for polynomials up to order $2N + 1$, while the GLL quadrature is exact for polynomials up to order $2N - 1$, where N is the DG order. To save the computational cost, we may approximate the mass matrix as a diagonal matrix through consistent mass lumping in a discrete formulation [17]. In the remaining of this chapter, we refer to the basis constructed on the GLL nodes as SEM basis, and the basis constructed on the GL nodes as GL basis.

To highlight the computational efficiency of the tensor product property, we take the procedure of computing $(\partial\varphi_{ijk}/\partial x, \varphi_{lmn})_{D^k}$ as an example. Consider the volume

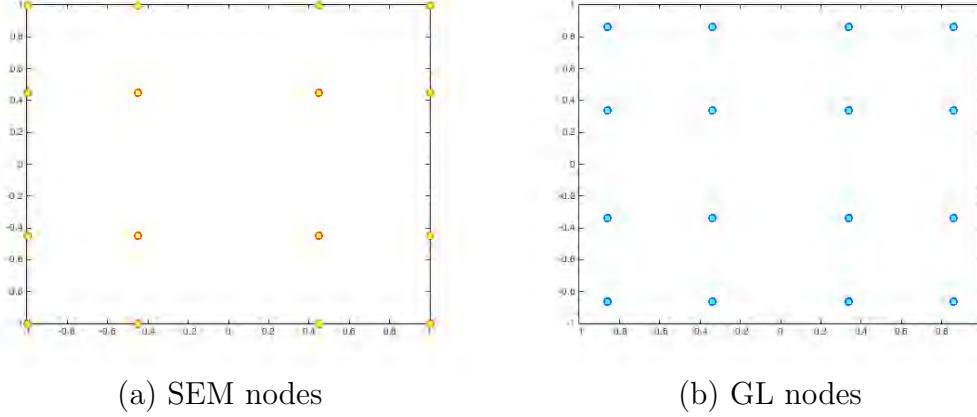


Figure 2.3: SEM and GL nodes: 2D example. (a) SEM (Gauss-Legendre-Lobatto) nodes contain the boundary nodes and are exact for polynomial integration up to order $2N - 1$ (N is the DG order); (b) GL (Gauss-Legendre) nodes are exact for polynomial integration up to order $2N + 1$.

integral

$$\begin{aligned}
 \left(\frac{\partial \varphi_{ijk}}{\partial x}, \varphi_{lmn} \right)_{D^k} &= \int_{D^k} \frac{\partial \varphi_{ijk}(x, y, z)}{\partial x} \varphi_{lmn}(x, y, z) dx dy dz \\
 &= \int_{\hat{D}} \frac{\partial \hat{\varphi}_{ijk}(r, s, t)}{\partial x} \hat{\varphi}_{lmn}(r, s, t) J(r, s, t) dr ds dt \\
 &= \int_{\hat{D}} \left(\frac{\partial \hat{\varphi}_{ijk}}{\partial r} \frac{\partial r}{\partial x} + \frac{\partial \hat{\varphi}_{ijk}}{\partial s} \frac{\partial s}{\partial x} + \frac{\partial \hat{\varphi}_{ijk}}{\partial t} \frac{\partial t}{\partial x} \right) \hat{\varphi}_{lmn} J dr ds dt,
 \end{aligned} \tag{2.20}$$

where \hat{D} is the reference element and J is the Jacobian of element D^k . We only expand the first term on the right-hand side of equation (2.20), and the other two terms can be derived analogously. Substituting $\hat{\varphi}_{ijk}$ and $\hat{\varphi}_{lmn}$ with definition (2.18), we have

$$\int_{\hat{D}} \frac{\partial \hat{\varphi}_{ijk}}{\partial r} \frac{\partial r}{\partial x} \hat{\varphi}_{lmn} J dr ds dt = \int_{\hat{D}} \frac{\partial \hat{l}_i(r)}{\partial r} \hat{l}_j(s) \hat{l}_k(t) \hat{l}_l(r) \hat{l}_m(s) \hat{l}_n(t) r_x J dr ds dt. \tag{2.21}$$

Applying quadrature rule to equation (2.21) yields

$$\begin{aligned}
\int_{\hat{D}} \frac{\partial \hat{l}_i(r)}{\partial r} \hat{l}_j(s) \hat{l}_k(t) \hat{l}_l(r) \hat{l}_m(s) \hat{l}_n(t) r_x J dr ds dt &\approx \sum_{\alpha=0}^N \sum_{\beta=0}^N \sum_{\gamma=0}^N \frac{\partial \hat{l}_i(r_\alpha)}{\partial r} \hat{l}_j(s_\beta) \hat{l}_k(t_\gamma) \hat{l}_l(r_\alpha) \hat{l}_m(s_\beta) \\
&\quad \hat{l}_n(t_\gamma) r_x(r_\alpha, s_\beta, t_\gamma) J(r_\alpha, s_\beta, t_\gamma) w_\alpha w_\beta w_\gamma \\
&= \frac{\partial \hat{l}_i(r_l)}{\partial r} r_x(r_l, s_m, t_n) J(r_l, s_m, t_n) \delta_{jm} \delta_{kn} w_l w_m w_n,
\end{aligned} \tag{2.22}$$

where N is the polynomial degree, (r_l, s_m, t_n) is the SEM/GL node corresponding to index (l, m, n) , and w_l, w_m, w_n are the quadrature weights.

With equation (2.22), the matrix-vector multiplication of the stiffness matrix becomes

$$\begin{aligned}
\sum_{i,j,k} \left(\frac{\partial \varphi_{ijk}}{\partial x}, \varphi_{lmn} \right)_{D^k} \hat{u}_{ijk} &= \left(\sum_{i=0}^N \frac{\partial \hat{l}_i(r_l)}{\partial r} r_x(r_l, s_m, t_n) \hat{u}_{ijk} + \sum_{j=0}^N \frac{\partial \hat{l}_i(s_m)}{\partial s} s_x(r_l, s_m, t_n) \hat{u}_{ijk} + \right. \\
&\quad \left. \sum_{k=0}^N \frac{\partial \hat{l}_i(t_k)}{\partial t} t_x(r_l, s_m, t_n) \hat{u}_{ijk} \right) J(r_l, s_m, t_n) w_l w_m w_n.
\end{aligned} \tag{2.23}$$

Operation (2.23), which corresponds to a single volume integral $\int_{D^k} \partial_x u_h \varphi_{lmn}$, has a computational cost of $\mathcal{O}(N)$ where N is the DG order. In contrast, such an operation has an $\mathcal{O}(N^d)$ ($d = 3$) cost on elements of other shapes. The total volume operation cost is $\mathcal{O}(N^{d+1})$ with tensor product, compared with $\mathcal{O}(N^{2d})$ without tensor product. We see that the hexahedral element is computationally cheap in terms of the volume integral.

2.3.2 Tetrahedron

The reference tetrahedron is mathematically represented as

$$tet_{ref} = \{(r, s, t) \mid -1 \leq r, s, t \leq 1; r + s + t \leq -1\}. \quad (2.24)$$

A reference tetrahedron is mapped to a straight-side tetrahedron with vertices $\mathbf{v}^1 \sim \mathbf{v}^4$ by

$$\mathbf{x} = -\frac{(1+r+s+t)}{2}\mathbf{v}^1 + \frac{(1+r)}{2}\mathbf{v}^2 + \frac{(1+s)}{2}\mathbf{v}^3 + \frac{(1+t)}{2}\mathbf{v}^4. \quad (2.25)$$

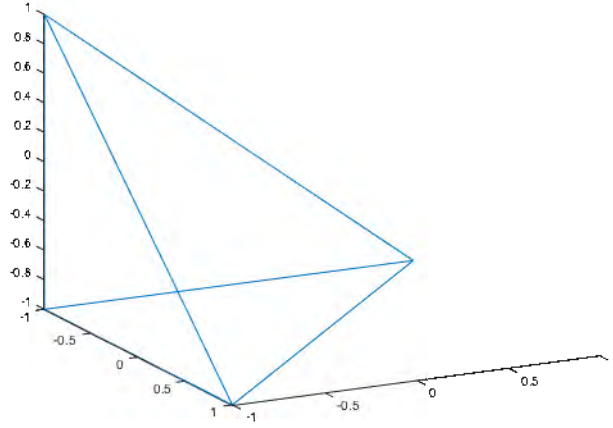


Figure 2.4: The tetrahedron reference element.

The determinant of the Jacobi matrix for an affine mapped (2.25) tetrahedron is constant, which reduces the implementation complexity and improves the computational efficiency.

Following the work in [47], we construct nodal basis functions each of which is evaluated as 1 on a given node and 0 on other nodal points. Because only a subset of the Lagrange basis functions is non-zero on the element faces, the nodal DG im-

plementation reduces the cost of calculating numerical fluxes. In addition, a series of nodal DG operators are given in [47], which is adopted in our implementation.

2.3.3 Prism

The reference prism is mathematically represented as

$$pri_{ref} = \{(r, s, t) \mid -1 \leq r, s, t \leq 1; r + t \leq 0\}. \quad (2.26)$$

A reference prism is mapped to a straight-side prism with vertices $\mathbf{v}^1 \sim \mathbf{v}^6$ by

$$\begin{aligned} \mathbf{x} = & -\frac{(r+t)(1-s)}{2} \frac{(1-s)}{2} \mathbf{v}^1 + \frac{(1-s)(1+t)}{2} \frac{(1+t)}{2} \mathbf{v}^2 + \frac{(1+r)(1-s)}{2} \frac{(1-s)}{2} \mathbf{v}^3 \\ & -\frac{(r+t)(1+s)}{2} \frac{(1+s)}{2} \mathbf{v}^4 + \frac{(1+s)(1+t)}{2} \frac{(1+t)}{2} \mathbf{v}^5 + \frac{(1+r)(1+s)}{2} \frac{(1+s)}{2} \mathbf{v}^6. \end{aligned} \quad (2.27)$$

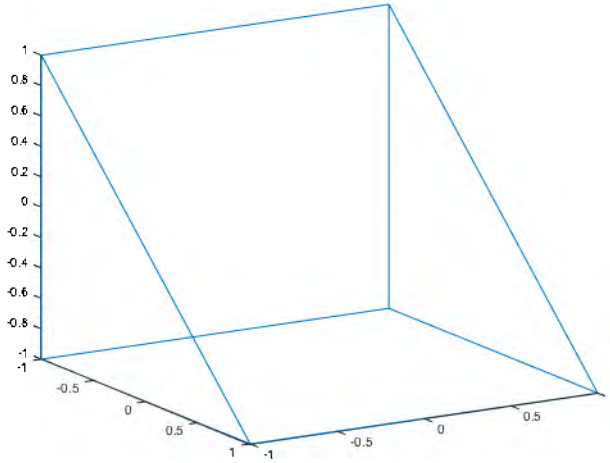


Figure 2.5: The prism reference element.

Due to the non-linearity of mapping (2.27), the mass matrix generated from a prismatic mesh is typically block diagonal but not strictly diagonal. To save the storage and avoid inverting the mass matrix in an explicit time-stepping scheme, the low-storage curvilinear DG (LSC-DG) introduced by Warburton [70, 71] is applied to

the prisms. The idea is dividing the original basis functions by the square root of the Jacobian and using these rational functions as the bases. Mathematically, the new basis functions are given by

$$\varphi_n^k = \frac{\tilde{\varphi}_n^k}{\sqrt{J^k}}, \quad (2.28)$$

where $\tilde{\varphi}_n^k$ is the polynomial basis function, and J^k is the Jacobian of element D^k . The inner product computed in the mass matrix is then

$$\begin{aligned} (\varphi_n^k, \varphi_m^k)_{D^k} &= \left(\frac{\tilde{\varphi}_n^k}{\sqrt{J^k}}, \frac{\tilde{\varphi}_m^k}{\sqrt{J^k}} \right)_{D^k} \\ &= \left(\frac{\hat{\varphi}_n^k}{\sqrt{J^k}}, \frac{\hat{\varphi}_m^k}{\sqrt{J^k}} J^k \right)_{\hat{D}} \\ &= (\hat{\varphi}_n^k, \hat{\varphi}_m^k)_{\hat{D}}, \end{aligned} \quad (2.29)$$

where $\hat{\varphi}_n^k$ are basis functions on reference element \hat{D} . If $\hat{\varphi}_n^k$, the reference basis functions, are orthonormal, the mass matrix becomes an identity which is convenient in the computation.

However, this convenience comes with a cost. First, taking derivatives becomes more complicated. For instance, calculating $\partial_x \varphi_n^k$ yields

$$\frac{\partial \varphi_n^k}{\partial x} = \frac{1}{\sqrt{J^k}} \frac{\partial \tilde{\varphi}_n^k}{\partial x} - \frac{\tilde{\varphi}_n^k}{2(J^k)^{3/2}} \frac{\partial J^k}{\partial x}. \quad (2.30)$$

Second, achieving the optimal order of convergence requires mesh regularities. For instance, one of the sufficient condition is given by

$$\|J^k\|_{L^\infty(D^k)} \left\| \frac{1}{J^k} \right\|_{L^\infty(D^k)} \leq C, \quad (2.31)$$

where C is a constant. Third, the integrals in the DG formulation cannot be evaluated exactly due to the rational basis functions. Noticing that the stability analysis

requests exact quadrature, we have to use a skew-symmetric formulation to guarantee stability. We apply the weak form formulation (2.8a) to the pressure field and the strong form formulation to the velocity field (2.9b), which yields

$$\int_{D^k} \frac{1}{\rho c^2} \frac{\partial p_h}{\partial t} q - \int_{D^k} \mathbf{v}_h \cdot \nabla q = - \int_{\partial D^k} \mathbf{v}_h^* \cdot \mathbf{n} q, \quad (2.32a)$$

$$\int_{D^k} \rho \frac{\partial v_{i,h}}{\partial t} w_i + \int_{D^k} \frac{\partial p}{\partial x_i} w_i = \int_{\partial D^k} (p_h - p_h^*) w_i n_i, \quad i = 1, 2, 3. \quad (2.32b)$$

Let $q = p_h$ and $\mathbf{w} = \mathbf{v}_h$, add these equations together, and substitute $\mathbf{v}_h^* = \{\{\mathbf{v}_h\}\} + \llbracket p_h \rrbracket / (2\rho c)$ and $p_h^* = \{\{p_h\}\} + \rho c \llbracket \mathbf{v}_h \rrbracket / 2$ into the equation. We then see the second terms on the left-hand side cancel out

$$\begin{aligned} \frac{1}{2} \frac{d}{dt} \int_{D^k} \left(\frac{1}{\rho c^2} p_h^2 + \rho \|\mathbf{v}_h\|_2^2 \right) &= - \int_{\partial D^k} \left(\frac{1}{\rho c} (p_h^-)^2 + \rho c (\mathbf{v}_h^- \cdot \mathbf{n})^2 - \frac{1}{\rho c} p_h^+ p_h^- \right. \\ &\quad \left. - \rho c (\mathbf{v}_h^+ \cdot \mathbf{n}) (\mathbf{v}_h^- \cdot \mathbf{n}) + p_h^+ (\mathbf{v}_h^- \cdot \mathbf{n}) + p_h^- (\mathbf{v}_h^+ \cdot \mathbf{n}) \right). \end{aligned} \quad (2.33)$$

Summing (2.33) over all elements, we obtain

$$\frac{d}{dt} \int_{\Omega} \left(\frac{1}{\rho c^2} p_h^2 + \rho \|\mathbf{v}_h\|_2^2 \right) = - \sum_k \int_{\partial D^k} \left((p_h^- - p_h^+)^2 / (\rho c) + \rho c (\mathbf{v}_h^- \cdot \mathbf{n} - \mathbf{v}_h^+ \cdot \mathbf{n})^2 \right). \quad (2.34)$$

If we denote $\mathcal{E} = \int_{\Omega} (p_h^2 / (\rho c^2) + \rho \|\mathbf{v}_h\|_2^2)$ as the energy, it is then straight-forward that $d\mathcal{E}/dt \leq 0$, and hence the energy does not grow with time.

2.3.4 Pyramid

The reference pyramid is mathematically represented as

$$pyr_{ref} = \{(r, s, t) \mid -1 \leq r, s, t \leq 1; r + t \leq 0; s + t \leq 0\}. \quad (2.35)$$

A reference pyramid is mapped to a straight-side pyramid with vertices $\mathbf{v}^1 \sim \mathbf{v}^5$ by

$$\begin{aligned} \mathbf{x} = & \frac{(r+t)}{2} \frac{(s+t)}{2} \frac{2}{(1-t)} \mathbf{v}^1 - \frac{(1+r)}{2} \frac{(s+t)}{2} \frac{2}{(1-t)} \mathbf{v}^2 \\ & + \frac{(1+r)}{2} \frac{(1+s)}{2} \frac{2}{(1-t)} \mathbf{v}^3 - \frac{(r+t)}{2} \frac{(1+s)}{2} \frac{2}{(1-t)} \mathbf{v}^4 + \frac{(1+t)}{2} \mathbf{v}^5. \end{aligned} \quad (2.36)$$

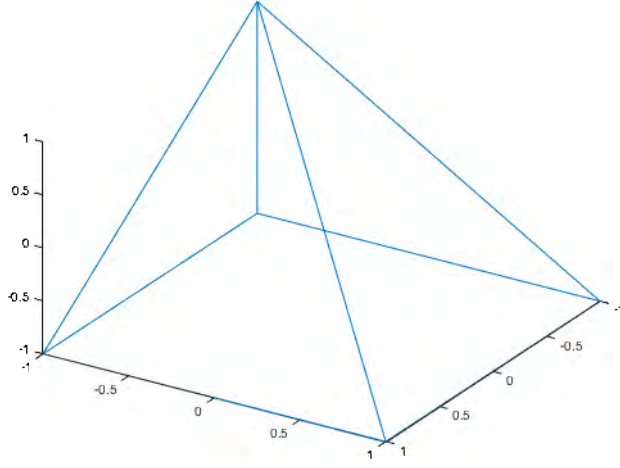


Figure 2.6: The pyramid reference element.

Bedrosian in his 1992 paper [66] observed that if polynomial interpolations are used as the basis functions for pyramidal elements, the Jacobian generated from the rational mapping (2.36) has a singularity, which leads to difficulties in the numerical quadrature. Instead of using many quadrature points, Bedrosian proposed a set of low order rational basis functions which resolved the issue [66]. After that, many researchers, including Bergot, Cohen, Durufle [67, 68], Chan and Warburton [69], extended Bedrosian's work to high order basis functions for pyramids. In this thesis, I adopt Chan and Warburton's work [69] which proposed a set of orthogonal bases for vertex-mapped pyramids. The mathematical formulas of the basis functions are

given by

$$\varphi_{i,j,k}(a, b, c) = l_i^k(a) l_j^k(b) \left(\frac{1-c}{2} \right)^k P_{N-k}^{2k+3,0}(c), \quad 0 \leq k \leq N, 0 \leq i, j \leq k, \quad (2.37)$$

where a, b, c are mapped by

$$a = \frac{2(1+r)}{1-t} - 1, \quad b = \frac{2(1+r)}{1-t} - 1, \quad c = t, \quad (2.38)$$

N is the DG order, l_i^k is the Lagrange polynomial interpolating Gauss-Legendre nodes, and $P_{N-k}^{2k+3,0}$ is the Jacobi polynomial.

The orthogonality of these bases results in a diagonal mass matrix, which can be easily inverted and save the computational storage.

2.4 Kernels and algorithms

Building on the numerical properties, I now address the implementation strategies for different element types.

In GPU programming, the procedures/functions running on a computational device are expressed as compute kernels. The DG solver is decomposed into several kernels, each of which is optimized for a specific task. For each element type, we construct the following three kernels

- **Volume kernel:** computes the contribution of the volume integrals (the terms integrated over the interior of D^k in the formulation).
- **Surface kernel:** computes the contribution of the surface integrals (the terms integrated over the element face ∂D^k in the formulation).
- **Update kernel:** performs a temporal update scheme of one time step to the

solution (third order Adams-Bashforth in our case), and interpolates the surface cubature points for next time step when necessary.

As the kernel performance determines the overall performance of the implementation, it is crucial to optimize the kernel code. The following techniques are employed for kernel tuning, which results in significant performance improvement.

- **Memory coalescing:** GPU threads access a chunk of memory through a memory bus, which indicates that multiple memory accesses are combined into a single transaction. To utilize the memory bus, the data layout is well-designed to guarantee coalesced memory access (i.e. continuous threads read/write contiguous memory locations).
- **Array padding:** In order to improve the performance, the solution array of each solution field is padded on each element to ensure that the array accesses are page aligned.
- **Kernel splitting:** Due to the limit of GPU shared memory on each core, the prism volume kernel is split into two parts. Although extra data loading is required when launching a new kernel, the overall performance is still improved by utilizing the shared memory [88].
- **Variable collection:** Some variables are collected together to reduce memory loading. For instance, quadrature weights can sometimes be embedded in geometric factors or Vandermonde matrices.
- **Element grouping:** The computational threads on a GPU are grouped into many blocks (see section 1.3). It is intuitive to let one block correspond to one element. However, grouping several elements into one block may utilize the execution cores and hence improve performance [49].

In the following sections, I will describe the kernel algorithms for each element type. Some kernels, which function similarly, have similar implementations, and hence may not be repeatedly listed.

2.4.1 Hexahedron

Exploiting the tensor product structure of hexahedral elements expresses the volume operations as 1D operators. The details are presented in algorithm 1.

Algorithm 1: Hexahedron volume kernel

Input: nodal value of solution $\mathbf{u} = (p, \mathbf{v})$, volume geometric factors $\partial(rst)/\partial(xyz)$, 1D derivative operator $D_{ij} = \partial \hat{l}_j / \partial x_i$, model parameters ρ, c

Output: volume contributions stored in array \mathbf{r}

- 1 **for** each element e **do**
- 2 **for** each volume node x_{ijk} **do**
- 3 Compute derivatives with respect to r, s, t

$$\frac{\partial \mathbf{u}}{\partial r} = \sum_{m=1}^{N+1} D_{im} \mathbf{u}_{mjk} \quad \frac{\partial \mathbf{u}}{\partial s} = \sum_{m=1}^{N+1} D_{jm} \mathbf{u}_{imk} \quad \frac{\partial \mathbf{u}}{\partial t} = \sum_{m=1}^{N+1} D_{km} \mathbf{u}_{ijm}$$
- 4 Apply chain rule to compute $\partial \mathbf{u} / \partial x, \partial \mathbf{u} / \partial y, \partial \mathbf{u} / \partial z$

$$\frac{\partial \mathbf{u}}{\partial x} = \frac{\partial \mathbf{u}}{\partial r} \frac{\partial r}{\partial x} + \frac{\partial \mathbf{u}}{\partial s} \frac{\partial s}{\partial x} + \frac{\partial \mathbf{u}}{\partial t} \frac{\partial t}{\partial x}$$
- 5 Calculate volume contributions
$$\mathbf{r}_p = -\rho c^2 \nabla \cdot \mathbf{v} \quad \mathbf{r}_v = -\frac{1}{\rho} \nabla p$$

The hexahedron surface kernels are different between SEM and GL implementations. Revisiting figure 2.3, we notice that the SEM nodes already contain the surface cubature points while the GL nodes do not. Therefore, the SEM implementation can utilize the nodal values to compute the numerical flux, while the GL implementation requires additional interpolations. Fortunately, these interpolations are 1D operations

and hence avoid significant computational costs. In algorithm 2, we present the procedure of the hexahedron surface kernel. In both implementations, the solution values on the surface cubature points are pre-computed and stored in array \mathbf{fQ} . The lines and variables marked with GL/SEM are the processes only needed by the GL/SEM implementation respectively.

Algorithm 2: Hexahedron surface kernel

Input: surface cubature values stored in \mathbf{fQ} , surface outward normal \mathbf{n} , lifting operator L (SEM), surface Vandermonde matrix V_f^{1D} (GL), inverse of mass matrix M^{-1} (GL), model parameters ρ, c

Output: surface contributions added to array \mathbf{r}

```

1 for each element  $e$  do
2   for each face  $f$  do
3     for each surface node  $x_{ij}$  do
4       Load  $\mathbf{u}^+$  and  $\mathbf{u}^-$  from  $\mathbf{fQ}$ , and compute numerical fluxes

           
$$\mathbf{pFlux} = -\frac{1}{2}\rho c^2 \left( \llbracket \mathbf{v} \rrbracket + \frac{1}{\{\{\rho c\}\}} \llbracket p \rrbracket \cdot \mathbf{n} \right)$$

           
$$\mathbf{vFlux} = -\frac{1}{2\rho c} \left( \llbracket p \rrbracket + \frac{\mathbf{n}}{\{\{\rho c\}\}} \llbracket \mathbf{v} \rrbracket \right)$$


5       (SEM) Apply  $L$  to the fluxes to obtain surface contributions

           
$$\mathbf{r}_p+ = L_{fij} \cdot \mathbf{pFlux} \quad \mathbf{r}_v+ = L_{fij} \cdot \mathbf{vFlux}$$


6     for each volume node  $x_{ijk}$  do
7       (GL) Apply lifting procedure with  $V_f^{1D}$  and  $M^{-1}$ 

           
$$\mathbf{r}_p+ = M_{ijk,ijk}^{-1} (V_f^{1D} \cdot \mathbf{pFlux}) \quad \mathbf{r}_v+ = M_{ijk,ijk}^{-1} (V_f^{1D} \cdot \mathbf{vFlux})$$


```

In the hexahedron update kernels, we need to pre-compute the surface information in array \mathbf{fQ} which will be used in the next time step. This makes the update kernels slightly different between the SEM and the GL implementations. Similar to the surface kernels, the GL implementation needs an extra interpolation while the SEM implementation does not. The pseudo code of hexahedron update kernel is listed in

algorithm 3.

Algorithm 3: Hexahedron update kernel

Input: right-hand side array \mathbf{r} , Adam-Bashforth coefficient c_i , surface Vandermonde matrix V_f^{1D} (GL)

Output: nodal value of solution \mathbf{u} , surface information \mathbf{fQ}

```

1 for each element  $e$  do
2   for each volume node  $x_n$  do
3     Update solution values using 3rd order Adam-Bashforth
                                     
$$\mathbf{u}(t_s) = \sum_{i=s-2}^s c_i \cdot \mathbf{r}(t_i)$$

4   for each surface node  $x_n$  do
5     (SEM) Store surface nodal value in  $\mathbf{fQ}$ 
                                     
$$\mathbf{fQ} = \mathbf{u}|_{\partial D^e}$$

6     (GL) Apply interpolation with  $V_f^{1D}$ , and fill  $\mathbf{fQ}$ 
                                     
$$\mathbf{fQ} = V_f^{1D} \mathbf{u}$$


```

2.4.2 Tetrahedron

The tetrahedron implementation benefits from the constant Jacobian which reduces the cost of loading geometric factors compared to other three element types [88]. In algorithm 4, I describe the tetrahedron volume kernel procedure, where some intermediate variables are introduced to accelerate the computation. In addition, the derivative operators, which are in fact the stiffness matrix left-multiplied by the inverse mass matrix, are given in [47].

The tetrahedron surface kernel is similar to the SEM hexahedron surface kernel. The only difference is in the lifting procedure. In SEM hexahedron, the lifting procedure is a point-wise operation, while in tetrahedron the lifting is an element-wise

Algorithm 4: Tetrahedron volume kernel

Input: nodal value of solution $\mathbf{u} = (p, \mathbf{v})$, volume geometric factors $\partial(rst)/\partial(xyz)$, derivative operator D_r, D_s, D_t , model parameters ρ, c

Output: volume contributions stored in array \mathbf{r}

```

1 for each element  $k$  do
2   for each volume node  $x_n$  do
3     Compute intermediate variables

      $\mathbf{VdotGr} = \mathbf{v} \cdot (r_x, r_y, r_z) \quad \mathbf{VdotGs} = \mathbf{v} \cdot (s_x, s_y, s_z) \quad \mathbf{VdotGt} = \mathbf{v} \cdot (t_x, t_y, t_z)$ 

4     Compute  $\nabla \cdot \mathbf{v}$  and  $p_r, p_s, p_t$ 

            $\nabla \cdot \mathbf{v} = D_r \cdot \mathbf{VdotGr} + D_s \cdot \mathbf{VdotGs} + D_t \cdot \mathbf{VdotGt}$ 

            $\frac{\partial p}{\partial r} = D_r \cdot p \quad \frac{\partial p}{\partial s} = D_s \cdot p \quad \frac{\partial p}{\partial t} = D_t \cdot p$ 

5     Apply chain rule to compute  $\partial p / \partial x, \partial p / \partial y, \partial p / \partial z$ 

            $\frac{\partial p}{\partial x} = \frac{\partial p}{\partial r} \frac{\partial r}{\partial x} + \frac{\partial p}{\partial s} \frac{\partial s}{\partial x} + \frac{\partial p}{\partial t} \frac{\partial t}{\partial x}$ 

6     Calculate volume contributions

            $\mathbf{r}_p = -\rho c^2 \nabla \cdot \mathbf{v} \quad \mathbf{r}_v = -\frac{1}{\rho} \nabla p$ 

```

operation, leading to an extra loop to process the tetrahedron lifting operation as listed in algorithm 5.

The tetrahedron update kernel is the same as the GL hexahedron update kernel except that the surface Vandermonde matrix is a 3D operator instead of a 1D matrix in GL hexahedron. Details are given in algorithm 6.

2.4.3 Prism

Due to the limited size of shared memory on a GPU, the prism volume kernel is split into two parts. Part 1 (algorithm 7) evaluates the solution values on quadrature points. Part 2 (algorithm 8) computes the inner product by multiplying the

Algorithm 5: Tetrahedron surface kernel

Input: surface cubature values stored in \mathbf{fQ} , surface outward normal \mathbf{n} , lifting matrix L , model parameters ρ, c

Output: surface contributions added to array \mathbf{r}

```

1 for each element  $k$  do
2   for each face  $f$  do
3     for each surface node  $x_m$  do
4       Load  $\mathbf{u}^+$  and  $\mathbf{u}^-$  from  $\mathbf{fQ}$ , and compute numerical fluxes

           
$$\mathbf{pFlux} = -\frac{1}{2}\rho c^2 \left( \llbracket \mathbf{v} \rrbracket + \frac{1}{\{\{\rho c\}\}} \llbracket p \rrbracket \cdot \mathbf{n} \right)$$

           
$$\mathbf{vFlux} = -\frac{1}{2\rho c} \left( \llbracket p \rrbracket + \frac{\mathbf{n}}{\{\{\rho c\}\}} \llbracket \mathbf{v} \rrbracket \right)$$


5     for each volume node  $x_n$  do
6       Apply  $L$  to the fluxes to obtain surface contributions

           
$$\mathbf{r}_p + = L \cdot \mathbf{pFlux} \quad \mathbf{r}_v + = L \cdot \mathbf{vFlux}$$


```

Vandermonde matrices.

The prism surface kernel differs from the tetrahedron surface kernel by the numerical flux formula and the lifting procedure. Algorithm 9 is the pseudo code of the prism surface kernel. Here \circ denotes the entry-wise multiplication.

The prism update kernel procedure, which is not listed here, is similar to the tetrahedron update kernel. Due to the LSC-DG algorithm, the basis functions for prisms are rationals, and one should be careful with the division of the square root of element Jacobian when evaluating solution values.

2.4.4 Pyramid

The pyramid volume kernel benefits from the constructions of the basis functions, which enables us to employ a series of semi-nodal operators to efficiently evaluate inner products in stiffness matrices. Further mathematical details are referred to

Algorithm 6: Tetrahedron update kernel

Input: right-hand side array \mathbf{r} , Adam-Bashforth coefficient c_i , surface Vandermonde matrix V_f)

Output: nodal value of solution \mathbf{u} , surface information \mathbf{fQ}

```

1 for each element  $k$  do
2   for each volume node  $x_n$  do
3     Update solution values using 3rd order Adam-Bashforth
        
$$\mathbf{u}(t_s) = \sum_{i=s-2}^s c_i \cdot \mathbf{r}(t_i)$$

4   for each surface node  $x_m$  do
5     Apply interpolation with  $V_f$ , and fill  $\mathbf{fQ}$ 
        
$$\mathbf{fQ} = V_f \mathbf{u}$$


```

Algorithm 7: Prism volume kernel part 1

Input: solution coefficients $\mathbf{u} = (p, \mathbf{v})$, volume geometric factors $\partial(rst)/\partial(xyz)$ and ∇J , (derivative) Vandermonde matrices V, V_r, V_s, V_t

Output: intermediate output stored in \mathbf{Qtemp}

```

1 for each element  $k$  do
2   for each cubature node  $x_n$  do
3     Evaluate weighted  $\nabla p$  and  $\mathbf{v}$  at cubature points
        
$$\mathbf{Qtemp}_p = w_n \nabla p(x_n) \quad \mathbf{Qtemp}_v = w_n \mathbf{v}(x_n)$$


```

[88]. The pseudo code is listed in algorithm 10.

The pyramid surface kernel is similar to the prism surface kernel, but the inverse of the mass matrix must be multiplied at the end of the surface kernel. The pyramid update kernel is the same as the tetrahedron update kernel.

Algorithm 8: Prism volume kernel part 2

Input: intermediate output stored in \mathbf{Qtemp} , transpose of (derivative)
 Vandermonde matrices V^T, V_r^T, V_s^T, V_t^T , model parameters ρ, c

Output: volume contribution stored in array \mathbf{r}

1 **for** each element k **do**

2 **for** each solution value u_n **do**

3 Compute inner product in stiffness matrices

$$\left(\mathbf{v}, \frac{\partial \varphi_n}{\partial x} \right)_{D^k} = V_x^T \mathbf{v} \quad \left(\frac{\partial p}{\partial x}, \varphi \right)_{D^k} = V^T \frac{\partial p}{\partial x}$$

4 Store volume contributions

$$\mathbf{r}_p = \rho c^2 (\mathbf{v}, \nabla \varphi_n) \quad \mathbf{r}_v = -\frac{1}{\rho} (\nabla p, \varphi_n)$$

2.5 Multi-rate time stepping

The kernels listed in section 2.4 correspond to the computation of one time step on one or a local group of elements, which is known as a local time step. Globally, however, different elements may be processed at different rates on the timeline. To address this issue, we introduce the multi-rate Adam-Bashforth method.

Early work of multi-rate time stepping for ordinary differential equations were discussed by Gear and Wells [72, 94, 95]. A detailed review of the multi-rate Adam-Bashforth method can be found in Stock's thesis [96], where 14 different strategies for the multi-rate method are introduced. The multi-rate scheme has been studied on electromagnetic fields [83, 97, 74], and it has been extended from Adam-Bashforth to Taylor expansion scheme [97] and Runge-Kutta scheme [74]. This section discusses the multi-rate third order Adam-Bashforth method on DG discretization.

Explicit time stepping is conditionally stable. To achieve stability, the Courant-Friedrichs-Lewy (CFL) condition must be satisfied. The CFL condition varies from element type to element type [88, 98], and the restriction of time step length is

Algorithm 9: Prism surface kernel

Input: surface cubature values stored in \mathbf{fQ} , surface outward normal \mathbf{n} , surface Vandermonde matrix V_f^T , surface quadrature weights \mathbf{w} , model parameters ρ, c

Output: surface contributions added to array \mathbf{r}

```

1 for each element  $k$  do
2   for each surface cubature node  $x_m$  do
3     Load  $\mathbf{u}^+$  and  $\mathbf{u}^-$  from  $\mathbf{fQ}$ , and compute numerical fluxes
        
$$\text{pFlux} = -\frac{1}{2}\rho c^2 \left( 2\{\{\mathbf{v}\}\} + \frac{1}{\{\{\rho c\}\}} \llbracket p \rrbracket \cdot \mathbf{n} \right)$$

        
$$\text{vFlux} = -\frac{1}{2\rho c} \left( \llbracket p \rrbracket + \frac{\mathbf{n}}{\{\{\rho c\}\}} \llbracket \mathbf{v} \rrbracket \right)$$

4   for each volume node  $x_n$  do
5     Apply  $V_f^T$  to the fluxes
        
$$\mathbf{r}_p + = V_f^T \cdot (\mathbf{w} \circ \text{pFlux}) \quad \mathbf{r}_v + = V_f^T \cdot (\mathbf{w} \circ \text{vFlux})$$


```

proportional to the element size

$$\Delta t^k \propto h^k, \quad (2.39)$$

where h^k is the characteristic length of element D^k . In a single-rate time stepping, we would have

$$\Delta t_{\min} = \min_k \Delta t^k, \quad (2.40)$$

which is a relatively small and inefficient time step length.

To relax restriction (2.40), we apply the multi-rate third order Adam-Bashforth method. In the multi-rate time stepping, elements are first grouped into levels according to their time step size Δt^k . Given N_{levels} groups, element D^k is grouped into

Algorithm 10: Pyramid volume kernel

Input: solution coefficients $\mathbf{u} = (p, \mathbf{v})$, volume geometric factors $\partial(rst)/\partial(xyz)$, derivative operators and their transposes $D_r, D_s, D_t, D_r^T, D_s^T, D_t^T$, model parameters ρ, c

Output: volume contributions stored in array \mathbf{r}

```

1 for each element  $k$  do
2   for each solution value  $u_n$  do
3     Compute intermediate variables

      $\mathbf{VdotGr} = \mathbf{v} \cdot (r_x, r_y, r_z) \quad \mathbf{VdotGs} = \mathbf{v} \cdot (s_x, s_y, s_z) \quad \mathbf{VdotGt} = \mathbf{v} \cdot (t_x, t_y, t_z)$ 

4     Compute inner products

      $(\mathbf{v}, \nabla \varphi_n) = D_r^T \cdot \mathbf{VdotGr} + D_s^T \cdot \mathbf{VdotGs} + D_t^T \cdot \mathbf{VdotGt}$ 

      $\left(\frac{\partial p}{\partial r}, \varphi_n\right) = D_r \cdot p \quad \left(\frac{\partial p}{\partial s}, \varphi_n\right) = D_s \cdot p \quad \left(\frac{\partial p}{\partial t}, \varphi_n\right) = D_t \cdot p$ 

5     Apply chain rule to compute  $(\nabla p, \varphi_n)$ 

      $\left(\frac{\partial p}{\partial x}, \varphi_n\right) = \left(\frac{\partial p}{\partial r}, \varphi_n\right) \frac{\partial r}{\partial x} + \left(\frac{\partial p}{\partial s}, \varphi_n\right) \frac{\partial s}{\partial x} + \left(\frac{\partial p}{\partial t}, \varphi_n\right) \frac{\partial t}{\partial x}$ 

6     Store volume contributions

      $\mathbf{r}_p = \rho c^2 (\mathbf{v}, \nabla \varphi_n) \quad \mathbf{r}_v = -\frac{1}{\rho} (\nabla p, \varphi_n)$ 

```

level l if its time step Δt^k satisfies

$$\begin{aligned}
 2^{N_{\text{levels}}-l} \Delta t_{\min} &\leq \Delta t^k < 2^{N_{\text{levels}}-l+1} \Delta t_{\min}, \quad l = 2, 3, \dots, N_{\text{levels}} \\
 2^{N_{\text{levels}}-l} \Delta t_{\min} &\leq \Delta t^k, \quad l = 1.
 \end{aligned} \tag{2.41}$$

In addition, we choose to guarantee that neighboring elements may have at most one level difference. We then allow each group to have its own time step length, and hence group of level l is integrated with a time step length of $\Delta t_l = 2^{l-1} \Delta t_{\min}$ on the timeline. The largest time step length, which is a global time step length, is defined by $\Delta t = \Delta t_1 = 2^{N_{\text{levels}}-1}$. An illustration of this grouping strategy is given in figure

2.7 and 2.8. Figure 2.7 groups the elements into three levels according to the element sizes. Figure 2.8 illustrates how elements are processed on timeline: level 1 elements are updated every Δt ; level 2 elements are updated every $\Delta t/2$; level 3 elements are updated every $\Delta t/4$.

However, different levels of elements are not totally independent of the others, and hence a communication strategy is needed. Due to the higher updating frequency on fine elements (high level), the information needed from their neighboring elements may not be available in the regular updating procedure. More specifically, when we update the fine elements (high level) on a fine time grid, we need the numerical fluxes from the coarse elements (low level) on a coarse time grid. For example, when we update level 2 elements at time $\Delta t/2$, information from level 1 elements at that time point is required. In order to provide that information from coarse elements to fine elements, the coarse-fine interfaces need to be updated on the fine time grid.

Taking level 1 and 2 elements for example again, we initially have the solution on both coarse and fine elements at time 0 (figure 2.9 (a)). We then update the fine elements, and the solution on fine grid is obtained at time $\Delta t/2$ (figure 2.9 (b)). At the same time, the coarse elements which have fine element neighbors need to update the solution on the coarse-fine interfaces at time $\Delta t/2$, which provides the numerical flux for updating the fine elements at next half time step (figure 2.9 (c)). Finally, we update fine elements from $\Delta t/2$ to Δt , and coarse elements from 0 to Δt , which completes a cycle of the global time step (figure 2.9 (d)).

Applying the third order Adam-Bashforth scheme, we obtain the update formula for l th level elements

$$\mathbf{u}^{n+\frac{k}{2^{l-1}}} = \mathbf{u}^{n+\frac{k-1}{2^{l-1}}} + \Delta t_l \sum_{s=1}^3 a_s \mathbf{r}^{n+\frac{k-s}{2^{l-1}}}, \quad (2.42)$$

where \mathbf{r} is the numerical contributions from the DG spatial discretization, n and k

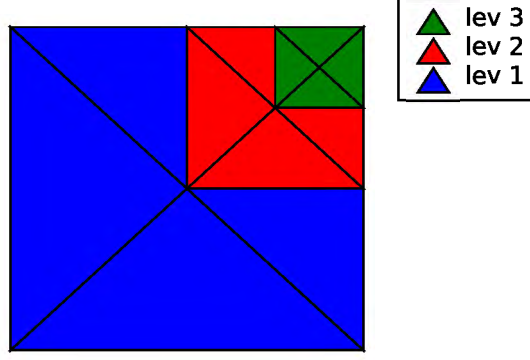


Figure 2.7: Illustration of multi-rate time stepping: elements are grouped into three levels.

are the global and local time-stepping indices respectively, and a_s are the Adam-Bashforth coefficients. To derive coefficients a_s , we first write down the semi-discrete form

$$\frac{d\mathbf{u}}{dt} = \mathbf{r}(t). \quad (2.43)$$

Without loss of generality, we take time step $\Delta t = 1$ and consider the solution value at $t = 1$. Integrating (2.43) from $t = 0$ to $t = 1$, we have

$$\mathbf{u}(1) - \mathbf{u}(0) = \int_0^1 \mathbf{r}(t) dt. \quad (2.44)$$

To discretize the right-hand side of (2.44), we take l_1, l_2, l_3 as Lagrange polynomials interpolating $0, -1, -2$, and interpolate \mathbf{r} at these three points, which yields

$$l_1 = \frac{1}{2}(t+1)(t+2), \quad l_2 = -t(t+2), \quad l_3 = \frac{1}{2}t(t+1), \quad \mathbf{r} = l_1 \mathbf{r}^0 + l_2 \mathbf{r}^{-1} + l_3 \mathbf{r}^{-2}. \quad (2.45)$$

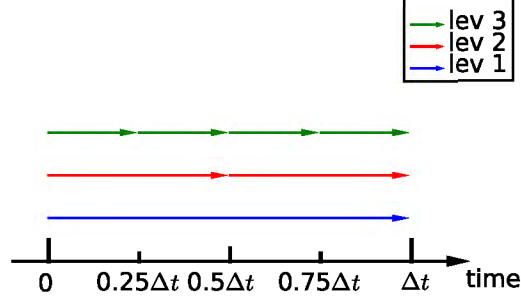


Figure 2.8: Illustration of multi-rate time stepping: level 1 elements are updated every Δt ; level 2 elements are updated every $\Delta t/2$; level 3 elements are updated every $\Delta t/4$.

Substitute (2.45) into (2.44)

$$\mathbf{u}(1) = \mathbf{u}(0) + \sum_{s=1}^3 \int_0^1 l_s(t) dt \mathbf{r}^{1-s} = \mathbf{u}(0) + \sum_{s=1}^3 a_s \mathbf{r}^{1-s}, \quad (2.46)$$

which results in

$$a_1 = \frac{23}{12}, \quad a_2 = -\frac{16}{12}, \quad a_3 = \frac{5}{12}. \quad (2.47)$$

Noticing that the numerical fluxes are required among different levels of elements, we need to compute the flux information from lower-level elements (coarser) passing to higher-level elements (finer) when the lower-level elements do not participate the local time stepping. To this end, the solution values on the coarse side of the coarse-fine interfaces are evaluated according to the following formula

$$\mathbf{u}^{n+\frac{k-1/2}{2^{l-1}}} = \mathbf{u}^{n+\frac{k-1}{2^{l-1}}} + \Delta t_l \sum_{s=1}^3 b_s \mathbf{r}^{n+\frac{k-s}{2^{l-1}}}, \quad (2.48)$$

with b_s corresponding to the half time step Adams-Bashforth coefficients. The derivation of b_s is similar to that of a_s . Instead of integrating from $t = 0$ to $t = 1$, we study

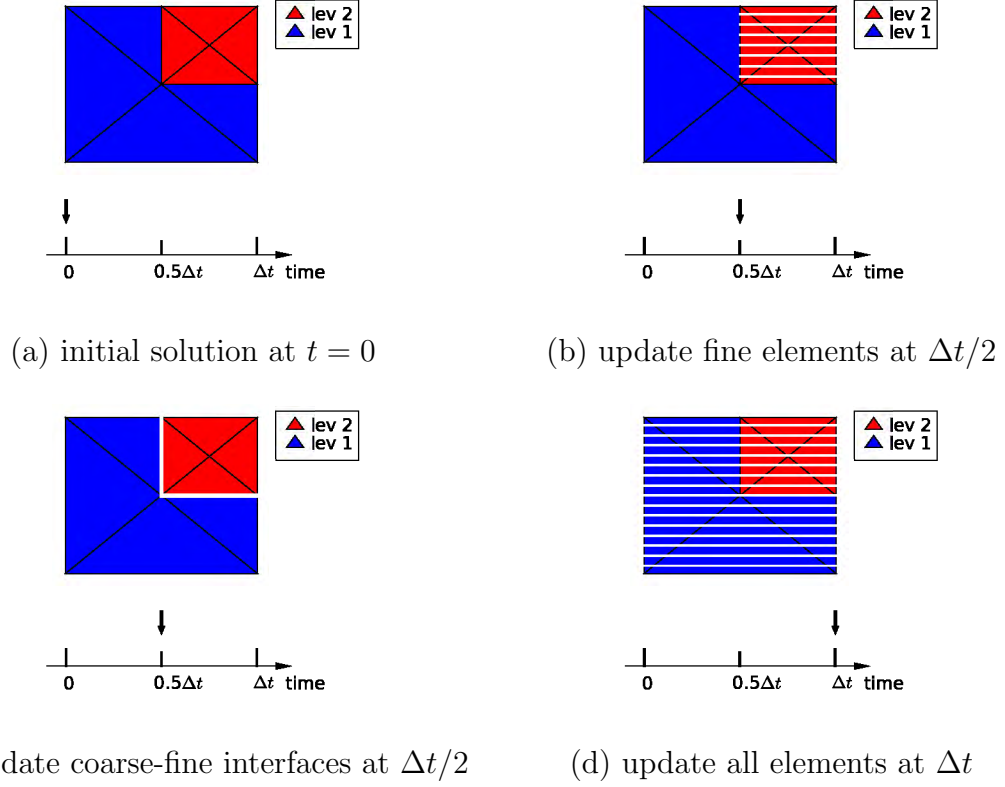


Figure 2.9: Illustration of multi-rate time stepping: fine and coarse elements are updated on fine and coarse time grids respectively. The white shaded areas are the updated places.

a half time step on $[0, 1/2]$. This gives us $b_s = \int_0^{1/2} l_s(t) dt$, and hence

$$b_1 = \frac{17}{24}, \quad b_2 = -\frac{7}{24}, \quad b_3 = \frac{2}{24}. \quad (2.49)$$

The multi-rate time stepping relaxes the global time step length restriction to several local restrictions. Elements of the different levels are updated at different rates, which reduces redundant computation and accelerates the implementation.

When the multi-rate time stepping is introduced, different elements have different workloads. Typically, the workload of an element at level l is proportional to 2^l , and is also related to the element type. This fact is an important issue needs to be addressed in parallelization. In addition, the communication among different levels of

elements must be addressed. The workload balancing along with other parallelization issues will be discussed in the next section.

2.6 Working on clusters

In section 2.4, we have discussed the kernels for a single GPU implementation, which is a common shared-memory system. In a shared-memory system, all computational threads can access the same memory. Therefore, threads can communicate by just reading and writing memories. For example, a GPU and a many-core CPU are common shared-memory systems. Although convenient in programming, the shared-memory system is usually limited by its computational capability for large-scale computations. In contrast, a distributed system has relatively independent computational nodes, and these nodes can communicate through message sending and receiving. To enable large-scale computation, a distributed-memory system, typically a cluster, should be used for running the DG solver.

In our implementation, we keep using OCCA [82] to work on shared-memory systems, but at the same time, we also apply Message Passing Interface (MPI) to let the shared-memory systems communicate with each other.

Similar to Gödel et al. [99], there are three levels of parallelism in our implementation (figure 2.10). First, we convert the topology of the computational mesh into a graph and send it to the graph partitioning software — METIS [100]. METIS then partitions the graph by taking the workload balancing and message passing efficiency into consideration. Using the output from METIS, we can partition the elements into multiple subdomains. Each subdomain is assigned to one MPI process corresponding to one shared-memory system (typically a GPU). Next, as each subdomain consists of many elements, one or several elements are assigned to a block on GPU. Finally, since one element has many degrees of freedom, each thread takes care of the computation

of one degree of freedom.

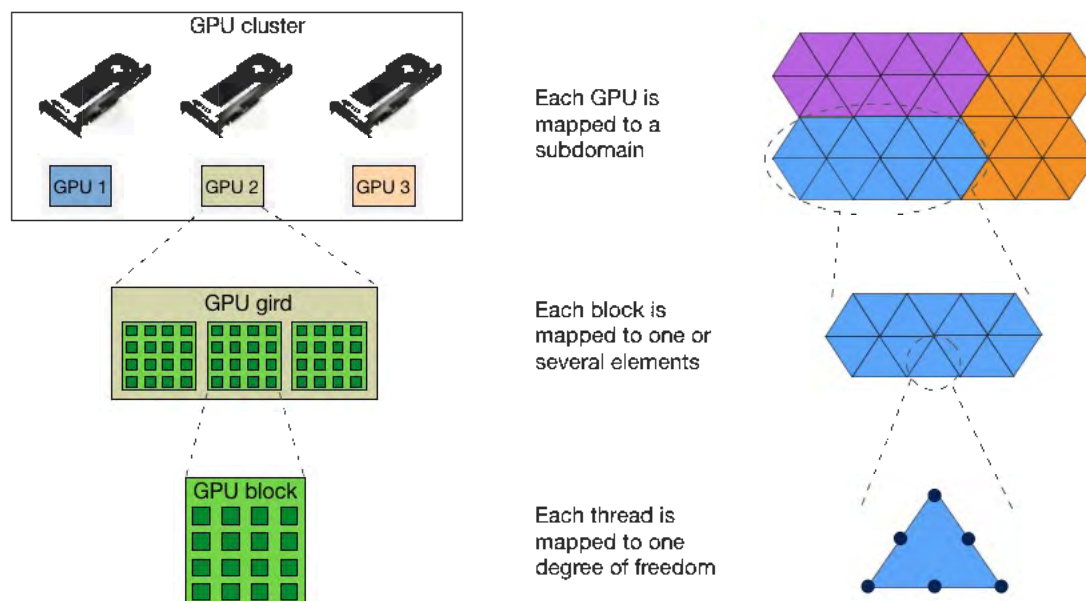


Figure 2.10: Illustration of 3-level parallelism: subdomains, elements, degrees of freedom are mapped to GPU grids, GPU blocks and threads respectively. Picture is reproduced from [99].

To assist the data movement, each partition of the domain is constructed with a layer of halo elements (figure 2.11). The halo elements only participate in the message passing process, but not the volume and surface computations. In the time stepping stage, the halo information is exchanged among the MPI processes at every time step. For each partition, the solution values on the partition interfaces, known as traces, are sent to the neighboring subdomains, and the trace information from neighboring subdomains is received and stored in the halo elements.

To hide the latency of the data transfer, we also divide one subdomain into inner elements and outer elements. The inner elements, as illustrated in figure 2.11, are the interior of the subdomain, and hence they do not require information from other MPI processes. The outer elements are the elements on the boundary of the subdomain, and they need to receive outside information to complete next updating cycle. In a

local time step, the inner elements process through volume, surface and update kernels without any barriers. In contrast, the outer elements cannot start the surface kernel until the MPI data transfer is finished in the current updating cycle. A workflow of this strategy is given in algorithm 11.

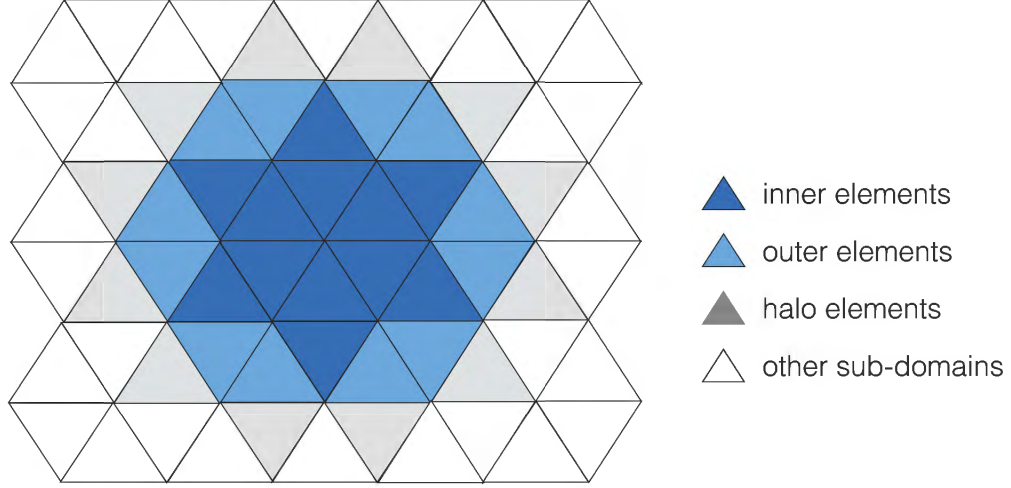


Figure 2.11: 2D illustration of MPI partition. The dark blue elements are inner elements; the light blue elements are outer elements; the gray elements are halo elements; the white elements are elements from other sub-domains.

Algorithm 11: Workflow of DG local time stepping

- 1 Start non-blocking MPI communication
 - 2 Call volume kernel for all elements
 - 3 Call surface kernel for inner elements
 - 4 Wait until MPI data transfer is finished
 - 5 Call surface kernel for outer elements
 - 6 Call update kernel for all elements
-

As the multi-rate time stepping has been introduced, the MPI communication becomes asymmetric among different levels of elements. The MPI data is transferred only when there is a need for the elements to update the right-hand side \mathbf{r} in equation (2.43). Therefore, sometimes the lower level elements send messages to higher level neighbors, but not the other way around. Taking the case where there are three

levels of elements as an example, we illustrate this technique in figure 2.12, where the elements in two different MPI processes are communicating, and the elements at the same heights are connected. At time 0, all elements compute the right-hand side, and hence everyone sends and receives messages; at time $\Delta t/4$ and $3\Delta t/4$, only level 3 elements compute the right-hand side, and hence the corresponding level 2 elements send messages; at time $\Delta t/2$, level 2 and 3 elements receive messages from the corresponding neighbors.

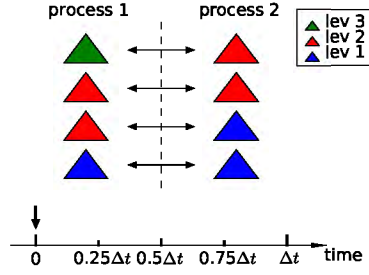
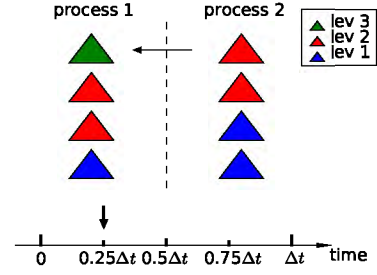
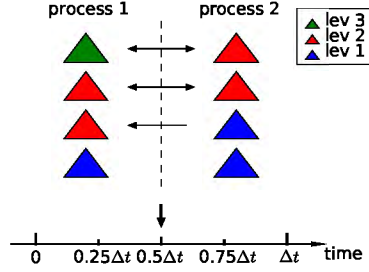
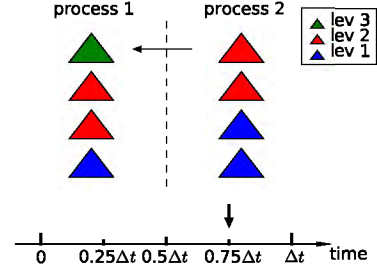
(a) all elements transfer data at $t = 0$ (b) level 3 elements receive data $\Delta t/4$ (c) level 2 and 3 elements receive data $\Delta t/2$ (d) level 3 elements receive data $3\Delta t/4$

Figure 2.12: Asymmetric MPI communication. Elements are distributed in two MPI processes, and the elements at the same heights are neighbors. At time $\Delta t/4, \Delta t/2, 3\Delta t/4$, lower level elements send messages to higher level elements, but not the other way around.

Due to the multi-rate time stepping, we notice that: (1) locally, the workload for different levels of elements become different; (2) globally, the communication frequen-

cies are different among different levels. To balance the workload, we pre-process the elements before partitioning the domain in METIS. In particular, we take two strategies suggested by Gödel et al. [83]: First, recalling that the elements are mapped to the graph nodes in METIS, we assign different nodes different weights in the graph partitioning. Given the element level l , the weight is proportional to 2^l ; Second, to reduce the MPI communication among fine elements, we lump the fine elements in order to remove the edges with heavy communication workload in the graph (i.e. multiple elements may be mapped to one node in the METIS graph).

2.7 Brief on data structure

Many of the operators and data structures of our codes are adopted from the nodal DG implementation [47]. However, to handle hybrid meshes, we present some additional data structures in this section.

2.7.1 Recognition of element types

Since different element types have different numbers of vertices, the element type information is stored in the element-to-vertex mapping array **EToV**. Array **EToV** is an array of size $K \times 8$, where K is the number of elements in a hybrid mesh and 8 is the maximum number of vertices of one element. Row k of **EToV** stores the vertex numbering of element k . For instance, a hexahedron has eight vertices, so a row in **EToV** containing eight non-zeros integers corresponds to a hexahedral element. Similarly, since a tetrahedron/pyramid/prism has four/five/six vertices, the corresponding rows have first four/five/six entries to be non-zero integers and zeros afterward.

An example of array **EToV** is given as follows,

$$\mathbf{EToV} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 5 & 6 & 9 & 8 & 7 & 10 & 0 & 0 \\ 1 & 2 & 3 & 4 & 11 & 0 & 0 & 0 \\ 3 & 4 & 11 & 12 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

where the corresponding mesh consists of four elements. The first row indicates a hexahedral element with vertices 1 ~ 8, and the second/third/fourth row corresponds to a prism/pyramid/tetrahedron.

To recognize the type of an element, one can simply check the number of non-zero entries in a corresponding row in array **EToV**. Listing 2.1 is a C++ sample code of recognizing element types.

```

1 elementType getElementType (int k) {
    if (EToV(k,5) == 0) return TET;
3   if (EToV(k,6) == 0) return PYR;
    if (EToV(k,7) == 0) return PRI;
5   return HEX;
    }

```

Listing 2.1: sample code of getting element types

2.7.2 Connectivity

The connectivity information is stored in two arrays **EToE** and **EToF**. Both arrays have sizes of $K \times 6$, where K is the number of elements in a hybrid mesh and 6 is the maximum number of faces of one element. Entry **EToE**(k, f) indicates that face f of element k is connected to element **EToE**(k, f), and **EToF**(k, f) is the corresponding face number of element **EToE**(k, f).

For example, the mesh connectivity in figure 2.13 can be represented as

$$\text{EToE} = \begin{pmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \quad \text{EToF} = \begin{pmatrix} 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \end{pmatrix}. \quad (2.50)$$

Therefore, we know face 3 of element 1 and face 5 of element 2 are connected.

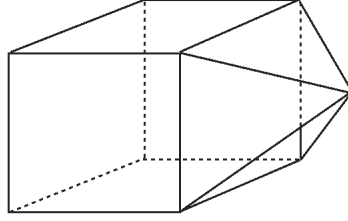


Figure 2.13: Illustration for data structure of mesh connectivity. A hexahedron and a pyramid are connected. The connectivity information is stored in **EToE** and **EToF** as given in (2.50).

2.7.3 Solution values

The solution values are stored in four separate arrays corresponding to four different element types. Similarly, the geometric factors, which include Jacobians, global coordinates, face normals and quadrature weights, are also split into four groups. When a computational kernel is executed, only the information needed for a given element type is loaded, which enables us to call specific kernels for different element types.

2.7.4 Traces of face values

To evaluate numerical fluxes in surface kernels, one needs the solution values on element faces which are also known as traces. At each time step, the trace information is pre-computed in the previous time step and stored in array **fQ**. Array **fQ** has the size of $K \times 6 \times (N + 1)^2 \times N_{\text{trace}}$, where K is the number of elements in a hybrid

mesh, 6 is the maximum number of faces of one element, $(N + 1)^2$ is the maximum degrees of freedom of one face, and N_{trace} is the number of fields needs to be stored. Although different elements have different numbers of faces and a triangular face has fewer degrees of freedom than a quadrangular face, we pad the trace storage for each element face into the same size in order to simplify the index searching procedure. Since the communication among elements is conducted through traces, a unified data container of trace information is efficient in computation.

To assist the index searching in computing numerical fluxes, an array `mapP` of size $K \times 6 \times (N + 1)^2$, which stores the node indices on adjacent faces, is pre-computed. In the surface kernel, each surface cubature point looks up the index of the overlapped point on its neighboring face in `mapP`, reads the corresponding values from `fQ`, and then computes the numerical fluxes.

2.8 Numerical convergence

In this section, we study the numerical convergence of the DG solver. As analyzed in [47], the guaranteed L^2 convergence rate of the DG method is $\mathcal{O}(h^{N+1/2})$ on general meshes, while the optimal convergence rate can be $\mathcal{O}(h^{N+1})$, where h is the characteristic size of the mesh elements and N is the polynomial degree in the DG approximation.

In the numerical experiments, I set the computational domain $\Omega = [-1, 1]^3$, set

the model parameters to constants by letting $c = \rho = 1$, and use exact solution of

$$\begin{aligned} p(x, y, z, t) &= \cos\left(\frac{\pi x}{2}\right) \cos\left(\frac{\pi y}{2}\right) \cos\left(\frac{\pi z}{2}\right) \cos\left(\frac{\sqrt{3}\pi t}{2}\right) \\ v_1(x, y, z, t) &= \frac{\sqrt{3}}{3} \sin\left(\frac{\pi x}{2}\right) \cos\left(\frac{\pi y}{2}\right) \cos\left(\frac{\pi z}{2}\right) \sin\left(\frac{\sqrt{3}\pi t}{2}\right) \\ v_2(x, y, z, t) &= \frac{\sqrt{3}}{3} \cos\left(\frac{\pi x}{2}\right) \sin\left(\frac{\pi y}{2}\right) \cos\left(\frac{\pi z}{2}\right) \sin\left(\frac{\sqrt{3}\pi t}{2}\right) \\ v_3(x, y, z, t) &= \frac{\sqrt{3}}{3} \cos\left(\frac{\pi x}{2}\right) \cos\left(\frac{\pi y}{2}\right) \sin\left(\frac{\pi z}{2}\right) \sin\left(\frac{\sqrt{3}\pi t}{2}\right) \end{aligned}$$

with Dirichlet boundary condition $p|_{\partial\Omega} = 0$ and $\mathbf{v}|_{\partial\Omega} = \mathbf{0}$ in time domain $t \in [0, 1]$. The numerical solution is initialized by L^2 projection. The time step length is set to be small enough so that the numerical error is dominated by the spacial discretization. The numerical experiments are run in double precision to guarantee enough accurate digits. Finally, the numerical error of pressure p is measured at $t = 1$ in L^2 norm using quadrature points that are sampled finer than the volume quadrature points employed in the DG volume integrations.

2.8.1 Verification on individual element types

The convergence study of each individual element type is conducted on a sequence of meshes. The mesh sequence is refined by repeating the patterns in figure 2.14, where each pattern is a cube containing one or several individual element types. The computational domain is made up of these cubic patterns in a uniform distribution. To make the test cases more general, we slightly perturb the inner vertices so that the meshes are no longer structured or affine. The perturbation is within $2\% \sim 5\%$ of the size of a cubic pattern. After the perturbation, the quadrangular faces may not be planar.

Table 2.1 \sim 2.4 list the L^2 errors and convergence rates of the DG solver on meshes of individual element types. As mentioned in section 2.3.1, we let the nodal points in

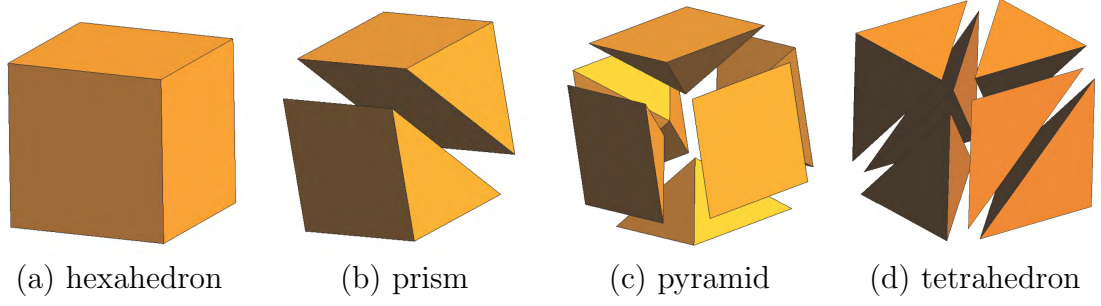


Figure 2.14: Patterns of mesh sequences. Each pattern is a cube consists of (a) 1 hexahedron/ (b) 2 prisms/ (c) 6 pyramids/ (d) 6 tetrahedra. Convergence of DG methods is tested on a sequence of meshes, which is refined by repeating the patterns.

hexahedral elements be either GLL points or GL points, which correspond to SEM and GL implementations respectively. In the tables, the numbers outside the round brackets are data from the SEM implementation while the numbers inside the round brackets are generated by the GL implementation.

In hexahedron, the difference between SEM and GL implementations affects both volume and surface cubatures; in prism and pyramid, the difference between these two implementations is the surface cubature on quadrangular faces; in tetrahedron, there is no difference between these two approaches.

order N	$N = 1$		$N = 2$		$N = 3$	
h	L^2 error	rate	L^2 error	rate	L^2 error	rate
0.5	6.1e-1 (1.5e-1)	-	3.8e-2 (2.1e-2)	-	2.5e-3 (2.0e-3)	-
0.25	2.4e-1 (4.0e-2)	1.35 (1.91)	4.6e-3 (2.6e-3)	3.05 (3.05)	2.0e-4 (1.3e-4)	3.64 (3.94)
0.125	6.9e-2 (9.8e-3)	1.80 (2.03)	6.8e-4 (3.2e-4)	2.76 (3.02)	1.5e-5 (8.1e-6)	3.74 (4.00)
0.0625	1.8e-2 (2.4e-3)	1.94 (2.03)	9.3e-5 (4.0e-5)	2.87 (3.00)	1.0e-6 (5.0e-7)	3.90 (4.02)

Table 2.1: Convergence study of hexahedral elements. Numbers outside the brackets correspond to the SEM approach, while the numbers inside the brackets are generated from the GL implementation.

order N	$N = 1$		$N = 2$		$N = 3$	
h	L^2 error	rate	L^2 error	rate	L^2 error	rate
0.5	3.2e-1 (2.0e-1)	-	4.0e-2 (3.5e-2)	-	6.4e-3 (6.1e-3)	-
0.25	9.3e-2 (4.9e-2)	1.78 (2.03)	5.2e-3 (4.4e-3)	2.94 (2.99)	3.9e-4 (3.7e-4)	4.03 (4.04)
0.125	2.6e-2 (1.2e-2)	1.84 (2.03)	7.1e-4 (5.6e-4)	2.87 (2.97)	2.7e-5 (2.3e-5)	3.85 (4.01)
0.0625	8.1e-3 (2.9e-3)	1.68 (2.05)	1.2e-4 (7.4e-5)	2.56 (2.92)	2.2e-6 (1.5e-6)	3.61 (3.94)

Table 2.2: Convergence study of prismatic elements. Numbers outside the brackets correspond to the SEM approach, while the numbers inside the brackets are generated from the GL implementation.

order N	$N = 1$		$N = 2$		$N = 3$	
h	L^2 error	rate	L^2 error	rate	L^2 error	rate
0.5	1.6e-1 (1.1e-1)	-	1.8e-2 (1.5e-2)	-	2.0e-3 (1.8e-3)	-
0.25	5.0e-2 (2.8e-2)	1.68 (1.97)	2.3e-3 (1.8e-3)	2.97 (3.06)	1.5e-4 (1.2e-4)	3.73 (3.91)
0.125	1.6e-2 (6.7e-3)	1.64 (2.06)	3.6e-4 (2.4e-4)	2.68 (2.91)	1.0e-5 (7.0e-6)	3.91 (4.01)
0.0625	5.0e-3 (1.6e-3)	1.68 (2.06)	5.5e-5 (3.0e-5)	2.71 (3.00)	8.2e-8 (4.3e-7)	3.61 (4.02)

Table 2.3: Convergence study of pyramidal elements. Numbers outside the brackets correspond to the SEM approach, while the numbers inside the brackets are generated from the GL implementation.

order N	$N = 1$		$N = 2$		$N = 3$	
h	L^2 error	rate	L^2 error	rate	L^2 error	rate
0.5	2.2e-1	-	5.1e-2	-	1.3e-2	-
0.25	5.6e-2	2.06	8.2e-3	3.27	6.8e-4	4.02
0.125	1.3e-2	2.08	7.7e-4	3.03	4.3e-5	3.98
0.0625	2.9e-3	2.16	9.6e-5	3.00	2.7e-6	3.99

Table 2.4: Convergence study of tetrahedral elements.

In almost all test cases, we observe the optimal convergence rates of $\mathcal{O}(h^{N+1})$ except some are polluted by the inexact SEM quadrature. Notice that the convergence rates can be harmed by the perturbation of the vertices, especially in LSC-DG

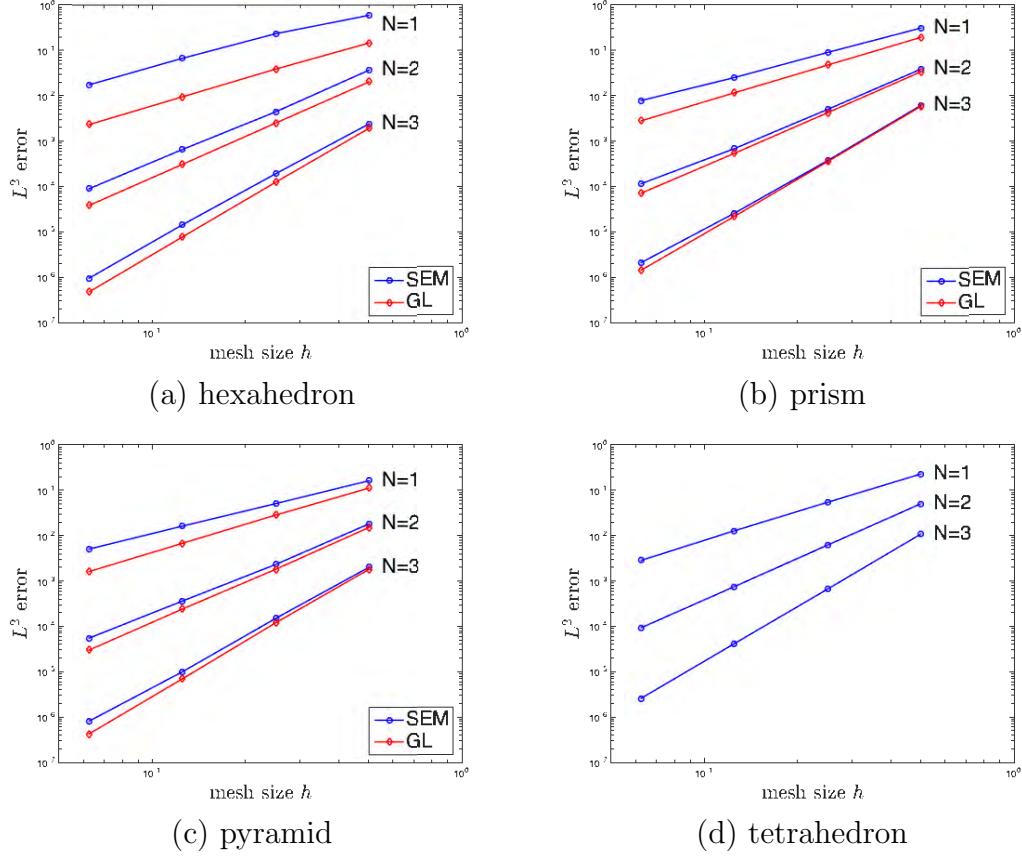


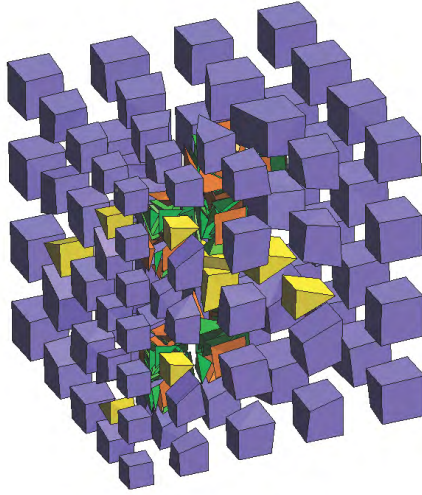
Figure 2.15: Convergence plots of individual element types. This figure reports the L^2 error versus the uniform mesh size h using a pure (a) hexahedron, (b) prism, (c) pyramid, and (d) tetrahedron element discretization of order $N = 1, 2, 3$. For the hexahedron, prism, and pyramid both SEM and GL performances are reported. Optimal convergence rates are observed in this figure. The GL implementation has smaller error than SEM implementation due to the accuracy of the quadrature.

for prisms (since LSC-DG require certain mesh regularities as discussed in section 2.3.3). However, in the above tests, the perturbation is small enough which makes no significant influence on the convergence. In addition, the GL approach has lower numerical errors and higher convergence rates than the SEM approach due to its relatively accurate quadrature rules.

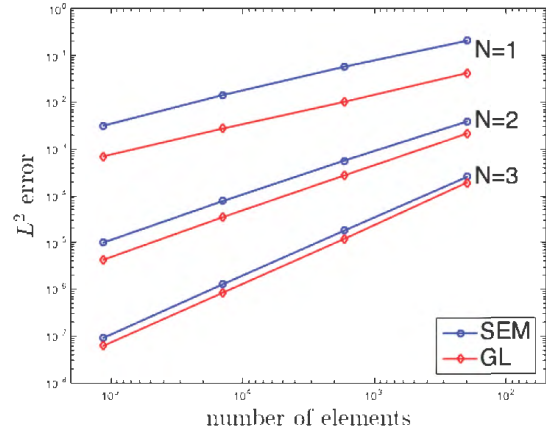
2.8.2 Verification on hybrid meshes

After convergence tests on individual element types, we now study the convergence on hybrid meshes which is a combination of all element types. Figure 2.16 (a) gives a sample of hybrid mesh used for tests, which is the coarsest mesh in the mesh sequence. The mesh sequence is generated by using **Gmsh**'s [101] “refine by splitting” function. As a result, each refinement approximately halved the mesh size. Statistics of the mesh sequence are reported in table 2.5, which lists the number of elements in each mesh.

Figure 2.16 (b) and table 2.6 report the numerical errors and convergence rates on hybrid meshes. The result of the convergence tests on hybrid meshes is similar to the study on meshes of individual element types: optimal convergence rates of $\mathcal{O}(h^{N+1})$ (N is the DG order) are observed in the experiments, and the GL implementation is slightly more accurate than the SEM implementation.



(a) hybrid mesh



(b) convergence plot

Figure 2.16: Hybrid mesh and convergence: (a) a sample of hybrid mesh with purple hexahedra, yellow prisms, orange pyramids and green tetrahedra; (b) convergence plot of a sequence of hybrid meshes. Both SEM and GL performances are reported for DG order $N = 1, 2, 3$.

In the above convergence studies, the numerical errors converge to zero when the mesh size decreases or the polynomial degree increases. The observed convergence rates are $\mathcal{O}(h^{N+1})$ as expected. The accuracy of the DG methods guarantees the reliability of the wave simulation in our applications. Following the convergence study, research on performance, which is crucial in DG applications, is carried out in the next section.

mesh	hexahedra	prisms	pyramids	tetrahedra	total elements
1	84	10	24	83	201
2	672	80	96	856	1704
3	5376	640	384	7616	14016
4	43008	5120	1536	64000	113664

Table 2.5: Configuration of hybrid meshes: number of elements in the mesh sequence for convergence study. Each refinement has approximately 8 times the number of elements in the previous mesh, and the mesh size is about halved.

order N	$N = 1$		$N = 2$		$N = 3$	
mesh	L^2 error	rate	L^2 error	rate	L^2 error	rate
1	2.0e-1 (4.1e-2)	-	3.8e-3 (2.1e-3)	-	2.5e-4 (1.9e-4)	-
2	5.6e-2 (1.0e-2)	1.83 (2.04)	5.6e-4 (2.7e-4)	2.76 (2.96)	1.8e-5 (1.2e-5)	3.80 (3.98)
3	1.4e-2 (2.7e-3)	2.00 (1.89)	7.7e-5 (3.5e-5)	2.86 (2.95)	1.3e-6 (8.6e-7)	3.79 (3.80)
4	3.1e-3 (6.9e-4)	2.17 (1.97)	1.0e-5 (4.3e-6)	2.95 (3.02)	9.3e-8 (6.3e-8)	3.81 (3.77)

Table 2.6: Convergence study of hybrid meshes. Numbers outside the brackets correspond to the SEM approach, while the numbers inside the brackets are generated from the GL implementation.

2.9 Performance study

In this section, we study the performance of our code on a single GPU by listing the estimated GFLOPS and bandwidth of the computational kernels and comparing the

time cost among them. The scalability of the code on multi-GPUs will be discussed in the future work. All the computation are run in CUDA mode on an Nvidia GTX 980 GPU using single precision. The specification of Nvidia GTX 980 is listed in table 2.7. In addition, since the performance is better for sufficiently large-size problems, the meshes for the performance tests are constructed with $K \approx 100,000$ elements.

Number of Cores	2,048
Core Clock	1,126 MHz
Boost Clock	1,216 MHz
Memory Clock	7.0 Gbps
Global Memory Size	4 GB
Shared Memory Size	48 KB
Peak Bandwidth	224 GB/s
Peak GFLOPS (Single Precision)	4,612

Table 2.7: Nvidia GTX 980 specification.

2.9.1 GFLOPS and Bandwidth of the kernels

GFLOPS, standing for giga floating-point operations per second, is a measure of computational performance. It quantifies the arithmetical efficiency (e.g. the rate of performing addition, subtraction, multiplication and division) of a computational implementation on a given hardware. Memory bandwidth, at the same time, measures the rate of data movement on the computational device. Specifically on a GPU, bandwidth measures how fast the computational kernels read and write the global memory, which is usually the main performance bottleneck.

Table 2.8 lists the GFLOPS and bandwidth of the volume, surface and update kernels. In hexahedron volume kernel, the GFLOPS is relatively low compared to volume kernels of other element types. However, the high bandwidth of the hexahedron volume kernels implies that the performance is bounded by IO (input/output). As discussed in [58], this limitation can be further improved by computing the geometric factors on the fly. GL-HEX surface and update kernels have higher GFLOPS than

SEM-HEX kernels, which is contributed by the additional linear interpolation in the GL-HHEX kernels. The prism kernels achieve relatively high performance in terms of GFLOPS due to the heavy use of `float4` variables. The arithmetic operations for `float4` variables are faster on certain GPUs [88]. The surface kernels of all element types have relatively low performance in terms of both GFLOPS and bandwidth, which is caused by the conditional statements and the non-coalesced access of trace data from neighboring elements.

order N	1	2	3	4	5	1	2	3	4	5
Hexahedron (SEM)										
Volume	201	268	343	397	468	166	167	175	171	175
Surface	50	48	66	60	54	86	86	122	116	109
Update	28	35	39	40	41	151	163	169	166	167
Hexahedron (GL)										
Surface	66	99	141	135	111	80	105	133	117	90
Update	84	104	116	121	124	153	163	169	167	166
Prism										
Volume	318	578	1200	1543	2004	158	118	107	77	59
Surface	119	249	534	381	719	74	76	84	35	41
Update	143	372	595	778	802	112	134	114	95	65
Pyramid										
Volume	137	268	349	374	416	99	92	62	38	26
Surface	106	238	681	477	841	71	87	138	59	66
Update	130	303	709	780	613	103	119	148	111	57
Tetrahedron										
Volume	237	424	474	672	740	122	99	58	49	34
Surface	105	175	389	530	749	86	83	109	95	88
Update	123	211	501	683	737	116	100	126	119	82

(a) GFLOPS

(b) bandwidth (GB/s)

Table 2.8: (a) GFLOPS and (b) bandwidth of the kernels. Each row corresponds to a kernel, and each column corresponds to a DG order. The numbers in the table are estimated by counting the floating-point operations/data movements on a single thread and then upscale to all threads.

2.9.2 Time cost comparison among different element types

We now compare the computational time cost among different element types. The computational time can be affected by many factors including mesh, polynomial degree and time step size. To be fair, we compare the time cost per degree of freedom

(DOF) in this study.

The experiments were conducted for each individual element type on meshes of $K \approx 10,000$ elements. The computational time was recorded for each kernel and then divided by the number of time steps and the number of degrees of freedom. Table 2.9 reports the time cost of different element types, where the reported time are scaled by the time cost of tetrahedron (i.e. the numbers in table 2.9 are the relative time cost to tetrahedron). Figure 2.17 is a histogram of total time cost for different element types, where the plotted values are also scaled by the time cost of tetrahedron.

order N	1	2	3	4	5
Hexahedron (SEM)					
Volume	1.1500	1.1007	0.6659	0.5864	0.4073
Surface	0.9851	0.8250	0.5982	0.5755	0.5048
Update	0.8668	0.6719	0.7552	0.7571	0.5141
Total	0.9688	0.8076	0.6623	0.6338	0.4790
Hexahedron (GL)					
Volume	1.1493	1.1010	0.6718	0.5910	0.4137
Surface	1.0900	0.7027	0.5784	0.5999	0.6570
Update	0.8628	0.6722	0.7613	0.7572	0.5159
Total	1.0288	0.7426	0.6564	0.6455	0.5359
Prism					
Volume	4.2523	3.6110	3.0179	3.1976	3.0633
Surface	1.1403	0.9972	0.9918	2.1673	1.5766
Update	1.2819	0.9022	1.2444	1.4472	1.4359
Total	1.5421	1.2950	1.5297	2.2222	1.9705
Pyramid					
Volume	2.0303	2.0848	1.9309	2.7446	2.8354
Surface	0.8043	0.9600	0.7290	1.4945	1.1896
Update	1.3149	0.9637	0.9481	1.2141	1.6319
Total	1.0987	1.1034	1.0696	1.7421	1.8321

Table 2.9: Time cost per degree of freedom (relative to tetrahedron) of hexahedron, prism and pyramid kernels. Each row corresponds to a kernel, and each column corresponds to a DG order.

From the results, we observe that the order of the time cost is hexahedron < tetrahedron < pyramid < prism. While the SEM-HEX implementation yields better performance than GL-HEX in high order cases, they have a similar time cost in

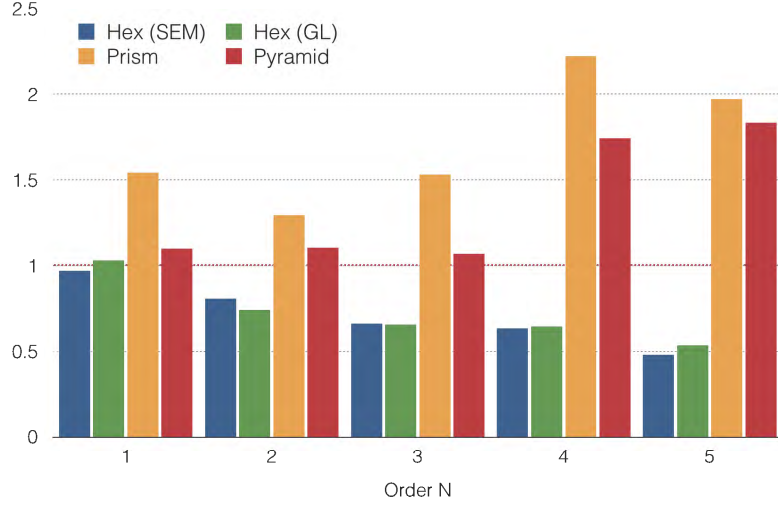


Figure 2.17: Total time cost per degree of freedom (relative to tetrahedron) for different element types. The time cost of tetrahedron is indicated by the red dash line of value 1. The relative time cost for hexahedron (both SEM and GL implementations), prism and pyramid are reported. The hexahedral elements are the most efficient elements especially at high orders.

general. Efficiency comparison between SEM-HEX and GL-HEX on CPUs was also reported by Kopriva and Gassner [102] with a similar conclusion. Both hexahedron implementations (SEM and GL) become more efficient than tetrahedron as the DG order increases. On the contrary, prism and pyramid are slower than tetrahedron, especially in high order cases. This slowness is caused by expensive derivative operations, loading of geometric node-wise factors, interpolation on surface points, kernel splitting and so on.

Although the comparison between a hybrid mesh and a pure tetrahedron mesh has not been studied, the result in this section suggested that a hex-dominant mesh has a high chance to beat a pure tetrahedron mesh in terms of performance.

2.10 Summary

This chapter introduces the formulation and implementation of the discontinuous Galerkin methods. The DG solver is built on hybrid meshes containing hexahedron, tetrahedron, prisms and pyramids. The properties of different element types are discussed in this chapter: for hexahedron, we take advantages of the tensor product property to highlight its computational efficiency [88]; for tetrahedron, we adopt the nodal DG implementation [47]; for prism, a low-storage curvilinear DG (LSC-DG) scheme is used to save memory storage [71]; for pyramid, orthogonal rational bases are employed to resolve the singularity issue on vertex-mapped pyramids [69].

The techniques to improve the performance of the DG solver are studied. We first exploit the implementation details of DG methods on a single GPU. Element-specific kernels are optimized to achieve the desired efficiency. Next, multi-rate time stepping is introduced to further accelerate the implementation. The multi-rate time stepping relaxes the global CFL condition into multiple local CFL constraints, which allows us to take large time step sizes for coarse zones in the mesh. Finally, the DG solver is parallelized on multiple GPUs to enable large-scale simulations. Workload assignment, asymmetric message passing, and latency hiding are discussed in this chapter.

Convergence and performance study of the DG implementation are presented at the end of this chapter. The convergence and the performance results meet our expectations. In general, we observe an L^2 error of $\mathcal{O}(h^{N+1})$ on different meshes where N is the DG order. Due to the 1D volume operations, hexahedral elements have the best performance in terms of time cost per degree of freedom, which encourages us to use hex-dominant meshes for large simulations.

Reverse time migration

Reverse time migration (RTM), early introduced in the 1980s [4, 8], is an imaging algorithm that requires solving two classes of wave equations. The first class, known as the forward problem, simulates the wave propagation with source signals. The second class solves the wave equation backward in time by injecting seismic data as external forces. RTM then takes the multiplication of these two wavefields and integrates it over time to evaluate an imaging condition. This process results in an image that pictures the subsurface structures.

RTM is attractive because it produces high-resolution images of subsurface layers, faults and salt bodies, but RTM is also challenging because imaging quality and computational efficiency must be addressed to obtain accurate images with affordable computational costs [103]. First, due to the classical RTM formulation, artifacts can be observed in the resulting images [103], and hence alternative imaging conditions have been proposed to improve the image quality. Yoon et al. [104] took advantages of Poynting vectors and designed a filter based on the directions of energy transport to suppress imaging artifacts. Liu et al. [105] decomposed the wavefield into downgoing and upgoing components and removed noise from the image by taking only the primary components. Modave et al. [9] proposed using characteristic fields instead of

pressure fields in the imaging condition and observed improved images. Zhang and Sun [106] suppressed RTM artifacts by proposing a modified version of forward equations. Second, RTM is a memory consuming algorithm, and many strategies were developed to save the memory cost. Symes [107] proposed the optimal checkpointing scheme to save the forward wavefields on a portion of the time grid, and re-initialize the forward solver in the backward stage using these stored wavefields. Clapp [108] suggested saving the boundary values of the forward wavefields and reproducing the forward wavefields at backward runs. Clapp [109] also proposed solving RTM with random boundaries to reduce memory requirement. Third, RTM is a computationally intensive algorithm, which leads to various high performance computing implementations. Different hardware accelerators may require different RTM implementations. In the last decade, the RTM algorithm has been studied on central processing units (CPUs) [110], graphics processing units (GPUs) [90, 111], and field programmable gate arrays (FPGAs) [112]. Each implementation claims superiority over others, but selecting the right hardware for RTM remains a topic of debate [113].

RTM is built on the numerical solutions to wave equations, and hence many numerical wave solvers have been studied on RTM. As we have already discussed in chapter 1, finite difference methods are one of the most popular schemes for RTM [16, 111], while pseudo-spectral methods [19] and finite element methods [60] have also been studied in the context of seismic imaging [114]. In recent years, the study of discontinuous Galerkin methods on RTM has been carried out by Modave et al. [9], which highlighted the strengths of DG-RTM with multi-GPU accelerations. Our DG-RTM code also adopts many implementation details from [9].

This chapter introduces reverse time migration and discusses the application of DG methods for RTM. In particular, I focus on the implementation of DG-RTM on many-core processors (i.e. GPUs).

3.1 Preliminaries

In order to accurately simulate wave propagation in the context of seismic exploration, the DG solver must be extended to meet two requirements: (1) the waves in real world typically propagate in long distance, and the domain of interest must be truncated; (2) the explosives in a seismic survey are modeled as point sources, and we must inject point sources into the numerical wave system. To this end, we combine the DG solver with perfectly matched layers (PML) and source injection with scattered-total field formulation. In this section, we discuss these techniques before working on RTM.

3.1.1 Perfectly matched layers

In practice, the domain of wave propagation is typically large. To save the computational cost, we truncate the physical domain to a region of interest with open boundaries. The wave energy must be absorbed near the computational domain boundary with the assumption that far-away wave interactions and reflections do not have influence on the region of interest. This energy absorption can be achieved by applying either absorbing boundary conditions or layer techniques. As a result, the simulation looks like a wave propagation in an unbounded domain. In this thesis, we adopt the perfectly matched layers (PML), which is one of the most widely used layer techniques.

The layer techniques extend the truncated domain with an artificial layer so that the wave energy is damped inside the layer. Perfectly matched layers, early introduced in 1994 by Berenger [115], are one class of the layer techniques and have drawn much attention of various researchers due to its dissipative and perfectly matched properties [116]. In other words, waves can transmit from non-PML region to PML region without reflection at the interface, and the PML layer absorbs waves without reflecting them back to the domain of interest. PML was first proposed for 2D

electromagnetic problems [115], and was quickly extended to three-dimensional and other wave-like problems [117, 118]. In this section, we focus on the PML formulation for 3D acoustic wave equations [116].

To apply PML, one needs to first pad the computational domain with an additional layer. As illustrated in figure 3.1, the center part is the regular computational domain while the perfectly matched layers are implemented near the boundary. We then solve the acoustic wave equation given by

$$\frac{\partial p(\mathbf{x}, t)}{\partial t} + \rho c^2 \nabla \cdot \mathbf{v}(\mathbf{x}, t) = - \sum_{i=1}^3 \sigma_i(\mathbf{x}) p_i(\mathbf{x}, t), \quad \mathbf{x} \in \Omega, \quad (3.1a)$$

$$\frac{\partial v_i(\mathbf{x}, t)}{\partial t} + \frac{1}{\rho} \frac{\partial p(\mathbf{x}, t)}{\partial x_i} = -\sigma_i(\mathbf{x}) v_i(\mathbf{x}, t), \quad i = 1, 2, 3, \quad \mathbf{x} \in \Omega, \quad (3.1b)$$

$$\frac{\partial p_i(\mathbf{x}, t)}{\partial t} + \rho c^2 \frac{\partial v_i(\mathbf{x}, t)}{\partial x_i} = -\sigma_i(\mathbf{x}) p_i(\mathbf{x}, t), \quad i = 1, 2, 3, \quad \mathbf{x} \in \Omega, \quad (3.1c)$$

$$p(\mathbf{x}, 0) = p_i(\mathbf{x}, 0) = v_i(\mathbf{x}, 0) = 0, \quad i = 1, 2, 3, \quad \mathbf{x} \in \Omega, \quad (3.1d)$$

where $\sigma_i(\mathbf{x})$ are the absorption coefficients that are zero in the non-PML region but a positive value inside the PML layer. Mathematically, $\sigma_i(\mathbf{x})$ must satisfy

$$\begin{cases} \sigma_i(\mathbf{x}) = 0, & \mathbf{x} \in \Omega_{\text{regular}}, \\ \sigma_i(\mathbf{x}) > 0, & \mathbf{x} \in \Omega_{\text{pml}}, \end{cases} \quad (3.2)$$

where $\Omega = \Omega_{\text{regular}} \cup \Omega_{\text{pml}}$. In the region where $\sigma_i(\mathbf{x}) = 0$ (i.e. non-PML region), equation (3.1) is simplified to a formulation without PML, and is equivalent to wave equation (2.1) as we have seen in chapter 2. The outer boundary of the PML layer is usually equipped with the first order absorbing boundary condition [116], which is given by

$$p - \rho c \mathbf{n} \cdot \mathbf{v} = 0, \quad (3.3)$$

where \mathbf{n} is a unit vector pointing at outward normal of the boundary.

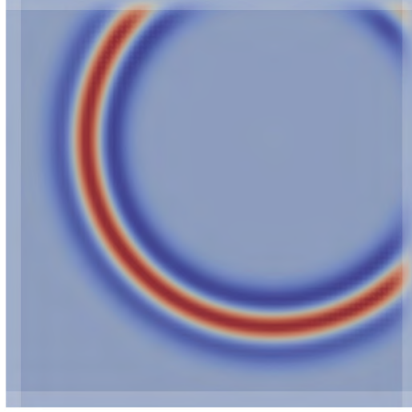


Figure 3.1: 2D illustration of PML. Computational domain is padded with additional layers. Waves are absorbed in the PML layers.

Due to linearity, we have $p = \sum_{i=1}^3 p_i$. The pressure field p is decomposed into three components, each of which corresponds to one Cartesian direction. Therefore, formulation (3.1) is also known as the split perfectly matched layers (SPML) [115]. The terms on the right-hand side of equation (3.1) are absorption terms which lead to an exponential decay of the system energy in time.

Numerical results in Modave et al. [119] suggest that the shifted hyperbolic function has a decent absorbing capability and is free of parameter tuning. The formula of the shifted hyperbolic function is given by

$$\sigma(x) = \alpha \frac{x}{\delta - x}, \quad (3.4)$$

where α is a positive parameter, δ is the thickness of the PML layer, and x is the distance from a point inside PML layer to the interface between non-PML domain and PML layer. This choice of absorption coefficient is adopted in our implementation.

3.1.2 Source term

In seismic imaging, the source signals generated by explosives or airguns are typically modeled as point sources. a point source is typically added as an external force. An intuitive way to add this term to the DG formulation is using L^2 projection. For instance, the source term is a Dirac delta function $\delta(\mathbf{x} - \mathbf{x}_0)$, and is approximated by $\delta(\mathbf{x} - \mathbf{x}_0) \approx \sum_n w_n \varphi_n^k(\mathbf{x})$, where $\varphi_n^k(\mathbf{x})$ are basis functions on element k and coefficients w_n are obtained by solving,

$$\int_{D_k} \varphi_m^k(\mathbf{x}) \left(\delta(\mathbf{x} - \mathbf{x}_0) - \sum_{n=1}^{N_p} w_n \varphi_n^k(\mathbf{x}) \right) d\mathbf{x} = 0, \quad \forall m = 1, \dots, N_p. \quad (3.5)$$

The Dirac delta function introduces singularity, thus destroying the high order accuracy in DG scheme. To resolve this issue, we introduce a scattered-total field formulation for source injection, which avoids dealing directly with the singularity in source injection [120].

3.1.2.1 Green's function

We begin the introduction to the scattered-total field formulation by first discussing Green's function. The Green's function in the second-order wave equation is the solution to

$$\frac{\partial^2 G(\mathbf{x}, t)}{\partial t^2} - c^2 \Delta G(\mathbf{x}, t) = \delta(t - s) \delta(\mathbf{x} - \mathbf{y}), \quad (3.6)$$

where $t, s \in \mathbb{R}^+$, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^3$, δ is the Dirac delta function, and c is the constant wave speed. The solution to this equation is

$$G(\mathbf{x}, t; \mathbf{y}, s) = \frac{1}{4\pi c^2 r} \delta(t - s - r_s/c) \quad (3.7)$$

with $r_s = \|\mathbf{x} - \mathbf{y}\|_2$ [121].

In order to solve the second-order wave equation

$$\frac{\partial^2 p(\mathbf{x}, t)}{\partial t^2} - c^2 \Delta p(\mathbf{x}, t) = f(t) \delta(\mathbf{x} - \mathbf{x}_0), \quad (3.8)$$

one can simply calculate the convolution of the Green's function and the source term, which yields

$$\begin{aligned} p(\mathbf{x}, t) &= \int_{\mathbb{R}^+} \int_{\mathbb{R}^3} f(s) \delta(\mathbf{y} - \mathbf{x}_0) G(\mathbf{x}, t; \mathbf{y}, s) d\mathbf{y} ds \\ &= \frac{1}{4\pi c^2 r} f(t - r/c) \end{aligned} \quad (3.9)$$

with $r = \|\mathbf{x} - \mathbf{x}_0\|_2$ [121].

Back to the first-order wave equation, the Green's function for the first-order wave system satisfies

$$\frac{\partial p}{\partial t} + \rho c^2 \nabla \mathbf{v} = F(t) \delta(\mathbf{x} - \mathbf{x}_0), \quad (3.10a)$$

$$\frac{\partial \mathbf{v}}{\partial t} + \frac{1}{\rho} \nabla p = 0, \quad (3.10b)$$

where $F(t) = \int_{-\infty}^t f(\tau) d\tau$ is the integrated source, and c and ρ are constant. The pressure solution (3.9) still holds for the first-order system (3.10). In addition, combining (3.9) and (3.10b) yields the velocity solution

$$\mathbf{v}(\mathbf{x}, t) = \frac{\mathbf{x}}{4\pi \rho c^2 r^2} \left(\frac{1}{r} F(t - r/c) + \frac{1}{c} f(t - r/c) \right). \quad (3.11)$$

Therefore, expressions (3.9) and (3.11) analytically give the solution to the acoustic wave equation in a free space with c and ρ being constant.

3.1.2.2 Scattered-total field formulation

To apply the scattered-total field formulation, we separate the wavefield into two fields: the incident field is the wavefield generated by the source; the scattered field is the wavefield without the contribution of the source injection. Since the wave equation is linear, the total wavefield is simply the summation of the incident and the scattered fields [120].

When the medium has constant velocity and density, the incident field is given by the analytic solution. Assuming that the media is homogeneous (i.e. constant velocity and density) near the source points, we partition the computational domain into two parts: the first part is a patch of elements containing the point source; the second part is the rest of the domain. The source is then injected through numerical fluxes.

Inside the patch (part 1), only the scattered field is computed. The exterior numerical flux (\mathbf{u}^+) passed into the patch contains only the information of the scattered field, which is obtained by subtracting the incident field (analytic solution) from the total field (numerical solution in part 2). Outside the patch (part 2), the total field is simulated. Therefore, the exterior numerical flux at the interface is a summation of the scattered field (numerical solution in part 1) and the incident field (analytic solution).

Denoting the domain of the total field as Ω_{tot} , the domain of the scattered field as Ω_{scatt} , and the interface of the total and the scattered fields as $\Gamma_{\text{int}} = \overline{\Omega_{\text{tot}}} \cap \overline{\Omega_{\text{scatt}}}$,

we can write the mathematical formulas for the total and scattered fields as

$$\frac{\partial p_{\text{tot}}(\mathbf{x}, t)}{\partial t} + \rho c^2 \nabla \cdot \mathbf{v}_{\text{tot}}(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \Omega_{\text{tot}}, \quad (3.12a)$$

$$\frac{\partial \mathbf{v}_{\text{tot}}(\mathbf{x}, t)}{\partial t} + \frac{1}{\rho} \nabla p_{\text{tot}}(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \Omega_{\text{tot}}, \quad (3.12b)$$

$$p_{\text{tot}}(\mathbf{x}, 0) = 0, \quad \mathbf{x} \in \Omega_{\text{tot}}, \quad (3.12c)$$

$$\mathbf{v}_{\text{tot}}(\mathbf{x}, 0) = \mathbf{0}, \quad \mathbf{x} \in \Omega_{\text{tot}}, \quad (3.12d)$$

$$p_{\text{tot}}(\mathbf{x}, t) = p_{\text{scatt}}(\mathbf{x}, t) + p_{\text{inc}}(\mathbf{x}, t), \quad \mathbf{x} \in \Gamma_{\text{int}},$$

$$(\text{OR}) \quad \mathbf{v}_{\text{tot}}(\mathbf{x}, t) \cdot \mathbf{n} = \mathbf{v}_{\text{scatt}}(\mathbf{x}, t) \cdot \mathbf{n} + \mathbf{v}_{\text{inc}}(\mathbf{x}, t) \cdot \mathbf{n}, \quad \mathbf{x} \in \Gamma_{\text{int}}, \quad (3.12e)$$

$$\frac{\partial p_{\text{scatt}}(\mathbf{x}, t)}{\partial t} + \rho c^2 \nabla \cdot \mathbf{v}_{\text{scatt}}(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \Omega_{\text{scatt}}, \quad (3.13a)$$

$$\frac{\partial \mathbf{v}_{\text{scatt}}(\mathbf{x}, t)}{\partial t} + \frac{1}{\rho} \nabla p_{\text{scatt}}(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \Omega_{\text{scatt}}, \quad (3.13b)$$

$$p_{\text{scatt}}(\mathbf{x}, 0) = 0, \quad \mathbf{x} \in \Omega_{\text{scatt}}, \quad (3.13c)$$

$$\mathbf{v}_{\text{scatt}}(\mathbf{x}, 0) = \mathbf{0}, \quad \mathbf{x} \in \Omega_{\text{scatt}}, \quad (3.13d)$$

$$p_{\text{scatt}}(\mathbf{x}, t) = p_{\text{tot}}(\mathbf{x}, t) - p_{\text{inc}}(\mathbf{x}, t), \quad \mathbf{x} \in \Gamma_{\text{int}},$$

$$(\text{OR}) \quad \mathbf{v}_{\text{scatt}}(\mathbf{x}, t) \cdot \mathbf{n} = \mathbf{v}_{\text{tot}}(\mathbf{x}, t) \cdot \mathbf{n} - \mathbf{v}_{\text{inc}}(\mathbf{x}, t) \cdot \mathbf{n}, \quad \mathbf{x} \in \Gamma_{\text{int}}, \quad (3.13e)$$

where p_{tot} and \mathbf{v}_{tot} are the total wavefields, p_{scatt} and $\mathbf{v}_{\text{scatt}}$ are the scattered wavefields, p_{inc} and \mathbf{v}_{inc} in (3.12e) and (3.13e) are the point source contribution given by (3.9) and (3.11).

The entire source injection procedure can be concluded as follows,

- Identify a patch of elements that contains the point source.
- Label elements in this patch as **scattered field**.

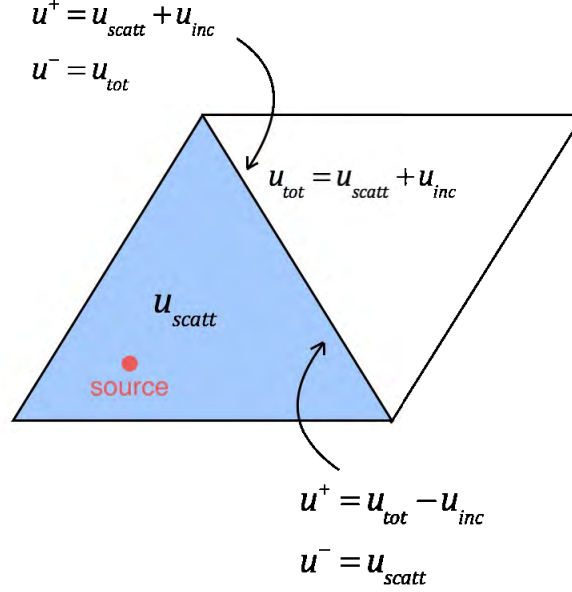


Figure 3.2: 2D illustration of source injection. The point source is injected through numerical fluxes. The blue element contains only the scattered field, while the white element has the total field simulation. The numerical flux is a function of the numerical solution \mathbf{u}_h and the analytic solution \mathbf{u}_{inc} .

- Provide exterior information in **scattered field** by $\mathbf{u}_{scatt}^+ = \mathbf{u}^+ - \mathbf{u}_{inc}$.
- Provide exterior information in the **halo of scattered field** by $\mathbf{u}_{tot}^+ = \mathbf{u}^+ + \mathbf{u}_{inc}$.

Figure 3.2 is also a 2D illustration for the partition of scattered-total field formulation, where the blue element belongs to the scattered field while the white element is inside the total field. In the simulation, source energy will be emitted from the blue-white interface to the total field.

3.2 Imaging condition

In reverse time migration, we solve the two types of acoustic wave equations to obtain two wavefields—the source wavefield p_S and the receiver wavefield p_R .

The source wavefield p_S , as indicated by its name, is generated by injecting a source into the domain. The source wavefield p_S is equipped with an zero initial

condition and an appropriate boundary condition (absorbing boundary condition in our case). More precisely, p_S satisfies,

$$\frac{\partial p_S(\mathbf{x}, t)}{\partial t} + \rho c^2 \nabla \cdot \mathbf{v}_S(\mathbf{x}, t) = f(t) \delta(\mathbf{x} - \mathbf{x}_S), \quad \mathbf{x} \in \Omega, \quad (3.14a)$$

$$\frac{\partial \mathbf{v}_S(\mathbf{x}, t)}{\partial t} + \frac{1}{\rho} \nabla p_S(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \Omega, \quad (3.14b)$$

$$p_S(\mathbf{x}, 0) = 0, \quad \mathbf{x} \in \Omega, \quad (3.14c)$$

$$\mathbf{v}_S(\mathbf{x}, 0) = \mathbf{0}, \quad \mathbf{x} \in \Omega, \quad (3.14d)$$

where ρ and c are the density and phase velocity of the media respectively, f is a signal depending on time t , δ is the Dirac delta function, \mathbf{x}_S is the location of the source. In our implementation, the boundary condition is an absorbing boundary given by the PML formulation discussed in section 3.1.1.

For the receiver wavefield p_R , instead of providing an initial condition, we give it a terminal condition at the end time of the wave propagation. Taking the terminal condition into consideration, we usually solve receiver wavefield p_R backward in time. Denoting $\tau = T - t$, with T indicating the end time of the propagation, we write the following formulation for p_R .

$$-\frac{\partial p_R}{\partial \tau} + \nabla \cdot \left(\frac{1}{\rho} \mathbf{v}_R \right) = \sum_{i=1}^{N_r} d(\tau) \delta(\mathbf{x} - \mathbf{x}_{R_i}), \quad \mathbf{x} \in \Omega, \quad (3.15a)$$

$$-\frac{\partial \mathbf{v}_R}{\partial \tau} + \nabla (\rho c^2 p_R) = 0, \quad \mathbf{x} \in \Omega, \quad (3.15b)$$

$$p_R(\mathbf{x}, \tau = 0) = 0, \quad \mathbf{x} \in \Omega, \quad (3.15c)$$

$$\mathbf{v}_R(\mathbf{x}, \tau = 0) = 0, \quad \mathbf{x} \in \Omega, \quad (3.15d)$$

where N_r is the number of receivers, \mathbf{x}_{R_i} is the location of the i 'th receiver, and $d(\tau)$ is the seismic data recorded at time $\tau = T - t$. The boundary condition in the implementation is the absorbing boundary which is same as the source wavefield.

Equation (3.14) and (3.15) can be solved by the DG scheme. With wavefields p_S and p_R , the classical imaging condition of RTM is given by

$$I(\mathbf{x}) = \sum_{s=1}^{N_s} \int_0^T p_S(\mathbf{x}, t; s) p_R(\mathbf{x}, t; s) dt, \quad (3.16)$$

where I is the imaging condition at location \mathbf{x} , s indicates shot s , and N_s is total number of shots. This imaging condition I provides an image of the subsurface structures and is the final output of RTM. As suggested in [9], the imaging condition can be improved by using

$$I(\mathbf{x}) = \sum_{s=1}^{N_s} \int_0^T q_S^-(\mathbf{x}, t; s) q_R^+(\mathbf{x}, t; s) dt, \quad (3.17)$$

where $q^\pm(\mathbf{x}, t; s) = p(\mathbf{x}, t; s) \pm c\rho \mathbf{e}_z \cdot \mathbf{v}(\mathbf{x}, t; s)$ are the characteristic fields, and \mathbf{e}_z is the unit vector pointing upwards. Since we are solving the first order wave equations, it is easy to extend (3.16) to (3.17) in terms of implementation. The comparison between the classical imaging condition and the imaging condition using characteristic fields was reported in [9]. In the following context, we will still use the notation in (3.16) if not otherwise specified.

An intuitive interpretation of imaging condition (3.16) is given in figure 3.3. For simplicity, we only consider the primary reflected wave here (the reflection caused by the first encountered reflector). The source wavefield (black solid) and receiver wavefield (blue dash) are depicted in the same computational domain. The two wavefields are non-zero near the reflector location at the same time. This gives a non-zero value of $I(\mathbf{x})$ at the reflector. In other locations, either source wavefield or receiver wavefield is zero at a given time, and hence $I(\mathbf{x})$ vanishes there.

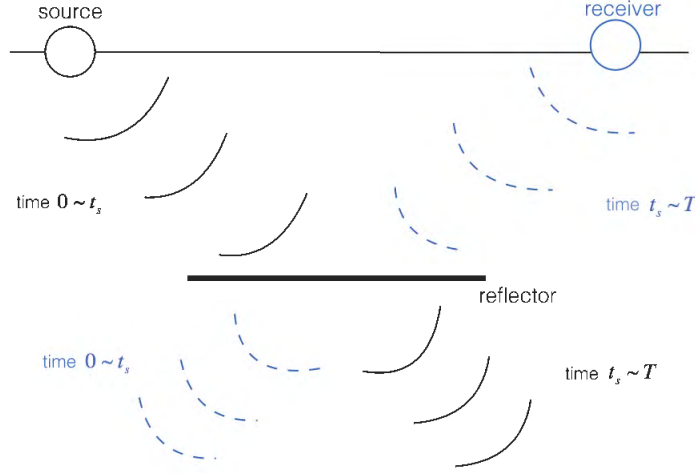


Figure 3.3: Interpretation of RTM. Only the wave front of the primary reflected wave is depicted. The black solid wavefield is the source wavefield, and the blue dash wavefield is the receiver wavefield. They have overlap at the reflector at time t_s , but not other places. As a result, expression (3.16) is non-zero at the reflector, but vanishes elsewhere.

3.3 Backward phase of the source wavefield

To compute the time integral (3.16), one has to have the knowledge of p_S and p_R at the same physical time. However, p_S is computed forward in time, while p_R is computed backward in time. This process prevents us from obtaining two wavefields at the same physical time. An intuitive way to deal with this issue is to compute the forward propagation first and save the wavefield p_S at all time points. At the backward propagation stage, the wavefield p_R is computed, and p_S is read from memory to evaluate the imaging condition.

This approach requires an excessive amount of storage and data movement, especially in three dimensions. Since our implementation runs on GPUs which have limited memory size, we want to avoid such data movement. Taking the high capability of floating point operations on GPUs into consideration, we could reduce the memory costs by increasing the number of floating point operations. This idea leads to two approaches: (1) The first one is the optimal checkpointing scheme proposed

by Symes [107], which saves the source wavefield p_S at a set of time checkpoints and uses the stored wavefield to initialize the forward wave propagation at the backward stage. This approach reduces memory cost by a factor logarithmic in the total number of steps but increases the computational complexity also by a factor logarithmic in the total number of steps; (2) As suggested in Clapp [108] and Modave [9], we can save the boundary values of the source wavefield p_S , and recover p_S backward in time at the backward stage. By computing the forward propagation twice, this approach requires storage only for boundary values but increases the computational complexity by a factor of 1.5. In this thesis, we adopt the second approach and present its details in the remaining of this section.

Similar to the implementation of traditional RTM, the entire computation is divided into two stages: the forward phase stage and the backward phase stage. In the forward phase stage, p_S and \mathbf{v}_S are numerically computed by solving equation (3.14). The boundary values of the pressure field at each time step and the terminal wavefield at $t = T$ are saved in this stage. Boundary values can also refer to values which lie on the interface between the perfectly matched layers (PML) and the non-PML region. In the backward phase stage, we solve both the receiver wavefield given by (3.15), and the backward phase of the source wavefield given by

$$-\frac{\partial p_S}{\partial \tau} + \rho c^2 \nabla \cdot \mathbf{v}_S = f(\tau) \delta(\mathbf{x} - \mathbf{x}_S), \quad \mathbf{x} \in \Omega, \quad (3.18a)$$

$$-\rho \frac{\partial \mathbf{v}_S}{\partial \tau} + \nabla p_S = 0, \quad \mathbf{x} \in \Omega, \quad (3.18b)$$

$$p_S(\mathbf{x}, \tau = 0) = \tilde{p}_S(\mathbf{x}, t = T), \quad \mathbf{x} \in \Omega, \quad (3.18c)$$

$$\mathbf{v}_S(\mathbf{x}, \tau = 0) = \tilde{\mathbf{v}}_S(\mathbf{x}, t = T), \quad \mathbf{x} \in \Omega, \quad (3.18d)$$

$$p_S(\mathbf{x}, \tau) = \tilde{p}_S(\mathbf{x}, \tau), \quad \mathbf{x} \in \Gamma, \quad (3.18e)$$

where $\tau = T - t$ is the reversed time variable, and \tilde{p}_S and $\tilde{\mathbf{v}}_S$ are data saved in the

forward phase.

It is trivial to show that the solutions of equation (3.14) and (3.18) are identical at the continuous level. Numerically, however, they are not the same unless an implicit time stepping scheme is used in the backward stage. Consider a simple evolution of forward Euler method,

$$\mathbf{u}^{n+1} = (I + \Delta t A) \mathbf{u}^n. \quad (3.19)$$

We must use the backward Euler method

$$\mathbf{u}^n = (I + \Delta t A)^{-1} \mathbf{u}^{n+1} \quad (3.20)$$

to obtain a discretely exact solution backward in time. More specifically in our case where the 3rd order Adam-Bashforth scheme is employed, the forward and backward schemes should be

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \sum_{i=0}^2 \alpha_i A \mathbf{u}^{n-i}, \quad (3.21)$$

$$\mathbf{u}^{n-2} = \frac{1}{\alpha_2} \left[- \sum_{i=0}^1 \alpha_i \mathbf{u}^{n-i} + A^{-1} \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} \right], \quad (3.22)$$

respectively, where α_i are AB3 coefficients. However, for the concern of computational efficiency, we still use an explicit time stepping at the backward stage. Due to this issue, additional numerical errors are introduced in the backward phase for the source wavefield. This error also brings difficulties when we want to further apply the imaging condition to full waveform inversion as we will discuss in chapter 4.

In brief, we recover the source wavefield at the backward phase stage without storing the entire source wavefield at a cost of increased computational time and additional numerical error.

3.4 Evaluation of the imaging condition

In order to obtain the imaging condition, we need to evaluate the time integral (3.16) numerically. In this section, we omit the dependence of the integrand on shot s in (3.16) for notational clarity. Thus, we want to numerically integrate

$$I(\mathbf{x}) = \int_0^T p_S(\mathbf{x}, t) p_R(\mathbf{x}, t) dt. \quad (3.23)$$

A common approach is to apply the trapezoidal rule. Equally dividing the interval $[0, T]$ into N_t sub-intervals with $0 = t_0 < t_1 < \dots < t_{N_t-1} < t_{N_t} = T$, one arrives at the trapezoidal rule,

$$\begin{aligned} I(\mathbf{x}) \approx & \frac{T}{2N_t} (p_S(\mathbf{x}, t_0)p_R(\mathbf{x}, t_0) + 2p_S(\mathbf{x}, t_1)p_R(\mathbf{x}, t_1) + \dots \\ & + 2p_S(\mathbf{x}, t_{N_t-1})p_R(\mathbf{x}, t_{N_t-1}) + p_S(\mathbf{x}, t_{N_t})p_R(\mathbf{x}, t_{N_t})). \end{aligned} \quad (3.24)$$

This approach gives an $\mathcal{O}(\Delta t^2)$ numerical error [122]. In our implementation, we adopt the quadrature scheme proposed by Modave et al. [9] to improve the accuracy of the time integration. The details of this quadrature scheme is given in the following of this section.

Assuming a 3rd order Adam-Bashforth scheme is used in time, we can represent the DG solution as

$$p(\mathbf{x}, t) = p(\mathbf{x}, t_{n-1}) + \Delta t \sum_{s=1}^3 \left(\int_0^{t^*(t)} l_s(t') dt' \right) r(\mathbf{x}, t_{n-s}), \quad (3.25)$$

where $t \in [t_{n-1}, t_n]$, $t^*(t) = (t - t_{n-1})/\Delta t$, $l_s(t)$ are Lagrange polynomials with interpolation nodes at 0, -1 and -2, and $r(\mathbf{x}, t)$ are the right-hand side vectors of the pressure field containing contributions from both volume and surface kernels as introduced in chapter 2.

Splitting the integral (3.23) as follows,

$$I(\mathbf{x}) = \sum_{n=1}^{N_t} \int_{t_{n-1}}^{t_n} p_S(\mathbf{x}, t) p_R(\mathbf{x}, t) dt, \quad (3.26)$$

we now need to evaluate the integral on each sub-interval. Substituting (3.25) into (3.26), we obtain

$$\begin{aligned} \int_{t_{n-1}}^{t_n} p_S(\mathbf{x}, t) p_R(\mathbf{x}, t) dt &= \Delta t p_S(\mathbf{x}, t_{n-1}) p_R(\mathbf{x}, t_{n-1}) \\ &+ \Delta t^2 \sum_{s=1}^3 a_s p_S(\mathbf{x}, t_{n-1}) r_R(\mathbf{x}, t_{n-s}) \\ &+ \Delta t^2 \sum_{s=1}^3 a_s p_R(\mathbf{x}, t_{n-1}) r_S(\mathbf{x}, t_{n-s}) \\ &+ \Delta t^3 \sum_{s_1=1}^3 \sum_{s_2=1}^3 C_{s_1 s_2} r_S(\mathbf{x}, t_{n-s_1}) r_R(\mathbf{x}, t_{n-s_2}). \end{aligned} \quad (3.27)$$

where $a_s = \int_0^1 \int_0^t l_s(t') dt' dt$, and $C_{s_1 s_2} = \int_0^1 \left(\int_0^t l_{s_1}(t') dt' \right) \left(\int_0^t l_{s_2}(t') dt' \right) dt$. These coefficients are given by

$$\mathbf{a} = \begin{pmatrix} 19/24 & -5/12 & 1/8 \end{pmatrix}', \quad (3.28a)$$

$$\mathbf{C} = \begin{pmatrix} 4703/5040 & -457/840 & 52/315 \\ -457/840 & 103/315 & -251/2520 \\ 52/315 & -251/2520 & 17/560 \end{pmatrix}. \quad (3.28b)$$

Scheme (3.27) has a numerical error of $\mathcal{O}(\Delta t^4)$ [9]. The additional data needed to evaluate this quadrature (i.e. r_S and r_R) is available in the DG update process and hence can be reused. The improvement of accuracy does not require a significant increase in cost compared with the trapezoidal rule.

3.5 Numerical results of synthetic surveys

We are now able to conduct synthetic surveys, in which the sources and receivers are artificially located in the computational domain, and the signals recorded by the receivers are generated by numerical simulations. Compared to realistic surveys where the signals are recorded with data acquisitions, synthetic surveys have better coherence between the velocity model and the recorded data, and hence produce better numerical results.

In this section, we will apply the RTM algorithm to three different models: (1) a single layer model, where there is only one reflector; (2) a multi-layer model, where there are multiple reflectors; (3) the Marmousi model, which is a widely used benchmark for many seismic applications [123].

3.5.1 Basic configurations

Before diving into the tests, we set some configurations that are applied to all the experiments in this section.

Throughout this section, we use the Ricker pulse as the source term. The Ricker pulse is given by

$$A(t) = (1 - 2\pi^2 f^2 (t - t_0)^2) e^{-\pi^2 f^2 (t - t_0)^2}, \quad (3.29)$$

where f is the peak frequency and t_0 is the time shift. An example of Ricker wavelet with $f = 5$ Hz and $t_0 = 0.2$ s is plotted in figure 3.4.

When we run the forward simulation, the receivers will record data. However, the direct wave, which is the wave transmits directly from the sources to the receivers, is recorded in the data. Since the direct wave does not contain any information from the subsurface, the data is muted in order to remove the direct wave. Figure 3.5 is an illustration of this muting procedure, where the data is taken from a one-shot

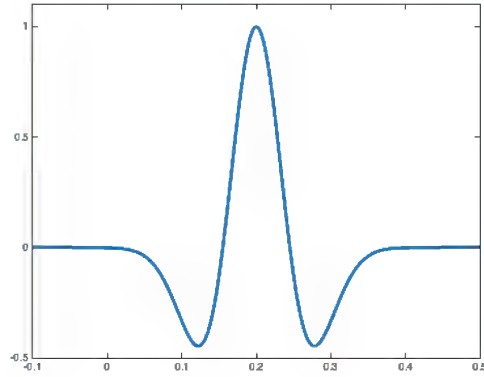
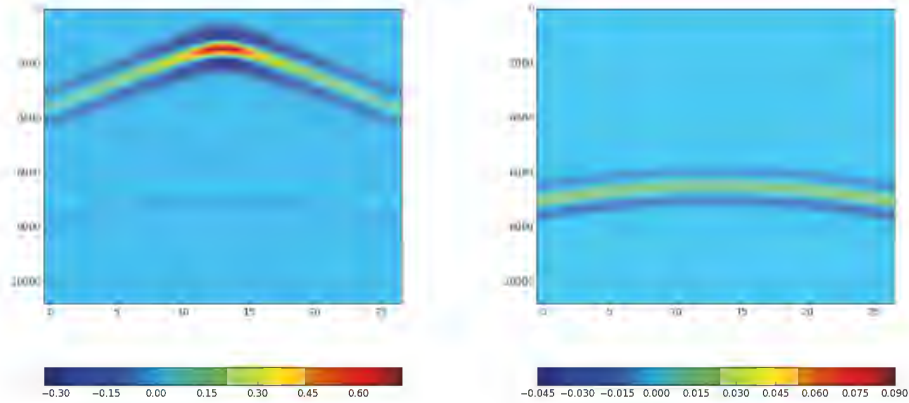


Figure 3.4: Example of Ricker wavelet, where the frequency $f = 5$ Hz and time shift $t_0 = 0.2$ s.

simulation on the single layer model. Figure 3.5 (a) shows a 2D slice of recorded data while figure 3.5 (b) shows an example of muted data. The data is muted by estimating the traveltimes from source to receiver and cutting the data before the estimated time point. As we can see, the direct wave in figure 3.5 (a) (data on the top) is removed in figure 3.5 (b).



(a) original data

(b) muted data

Figure 3.5: Data muting in RTM configuration: (a) 2D slice of the original recorded data; (b) 2D sliced of muted data, where direct waves are removed.

Finally, absorbing boundaries (i.e. PML) are applied to all the boundaries in the

following experiments.

3.5.2 Single layer model

We start with the simplest model, where there is only a single layer in the model. The solution should ideally give us the information about the layer interface in the RTM image.

The configuration is: the computational domain is a cube of size $[0 \text{ km}, 2 \text{ km}]^3$; a velocity interface is located at $z = 1.0 \text{ km}$; the top part of the model has a velocity of 1 km/s , and the bottom part has a velocity of 2 km/s (figure 3.6); 100 Ricker sources of 6 Hz peak frequency are distributed evenly on plane $z = 0.21 \text{ km}$, and 729 receivers are evenly distributed on plane $z = 0.3 \text{ km}$ (figure 3.7); recording end time is $T = 3.0 \text{ s}$; the simulation is executed on a pure hexahedral mesh using GL-HEX algorithms as discussed in the previous chapter.

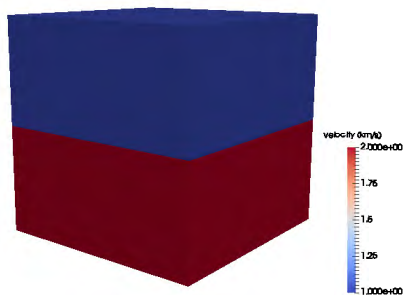
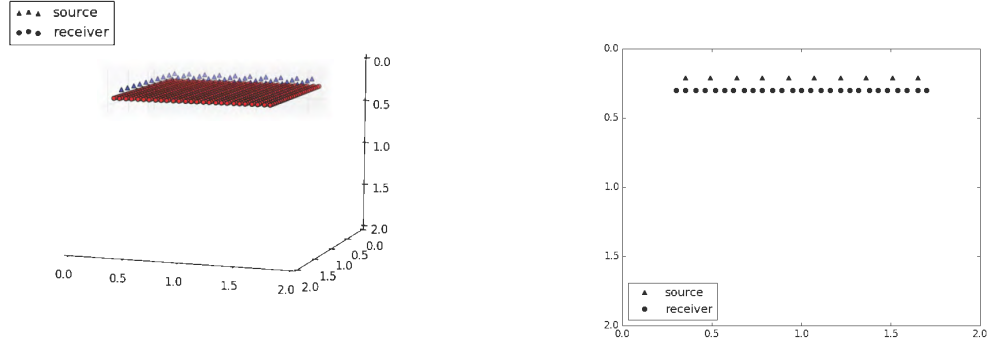


Figure 3.6: Single layer velocity model, where the top has 1 km/s velocity and the bottom has 2 km/s velocity.



(a) 3D view of the source and receiver locations (b) 2D slice of the configuration

Figure 3.7: Single layer example: sources and receivers, which are located at the top of the domain.

The results of this experiment are reported in figure 3.8. In the image, we observe a line segment at the reflector location, which suggests that the layer interface is successfully identified by the RTM procedure.

The image quality is affected by the frequency of the signal. Because the frequency is inversely proportional to the wavelength, higher frequency would lead to smaller wavelength, and smaller wavelength may reveal higher resolution of the subsurface structure. For instance, the line segment in figure 3.8 is wide in z -direction, and increasing wave frequency may reduce the width of this line segment which makes the image sharper. However, simulating high frequency waves requires finer meshes or higher order polynomials in DG to guarantee the fidelity of the wave simulation. The computational cost increases as we must have enough degrees of freedom allocated within one wavelength.

3.5.3 Multi-layer model

To further validate the algorithm and our implementation, we test a more complicated model where there are several reflectors. To simplify the computation and reduce the

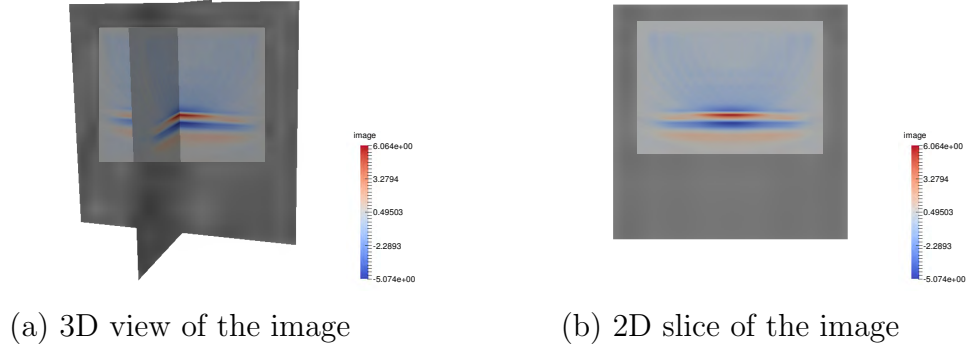
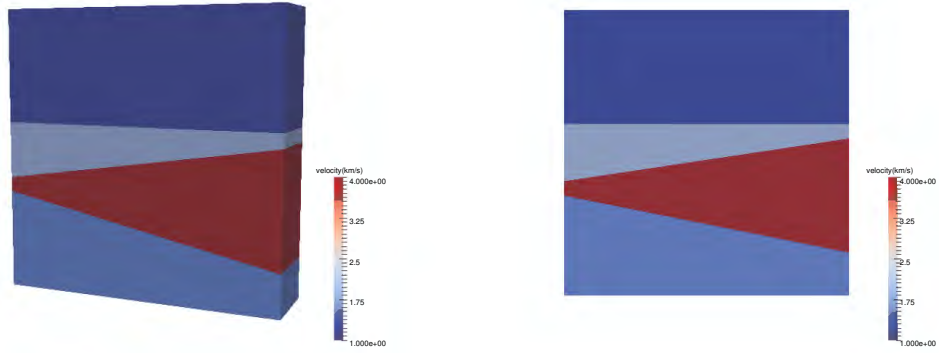


Figure 3.8: Single layer example: results of the RTM algorithm. The layer is revealed in the RTM image.

computational cost, we set this test case to be a 2.5D model, where the dimension along y -axis is short in distance and a constant in velocity model.

The configuration is: the computational domain is a cube of size $[0 \text{ km}, 2 \text{ km}] \times [0 \text{ km}, 0.6 \text{ km}] \times [0 \text{ km}, 2 \text{ km}]$; multiple layers are located in the model, and their velocities are (from top to bottom) 1 km/s, 2 km/s, 4 km/s and 1.5 km/s respectively (figure 3.9); 9 Ricker sources of 6 Hz peak frequency are distributed evenly on the line $y = 0.2 \text{ km}, z = 0.3 \text{ km}$, and 25 receivers are evenly distributed on the line $y = 0.2 \text{ km}, z = 0.4 \text{ km}$ (figure 3.10); recording end time is $T = 3.0 \text{ s}$; the simulation is carried out on a pure tetrahedral mesh.

The results of the RTM algorithm are reported in figure 3.11. As the wave transmits deep into the domain, the wave energy is damped. The data signal which conveys the information of the lower reflectors is weak in terms of magnitude. As a result, the lower reflectors are less significant in the RTM image than the upper structures. To resolve this issue, some filtering techniques are proposed to improve the quality of the image. For instance, Laplacian filtering applies Laplacian operator to images in order to highlight regions of rapid intensity change [124], and Q-filtering technique compensates the amplitude attenuation of waves to enhance the resolution of images [125]. For simplicity, we just plot the image in different color scales to observe the



(a) 3D view of the velocity model

(b) 2D view of the velocity model

Figure 3.9: Multi-layer velocity model. Velocities are (from top to bottom) 1 km/s, 2 km/s, 4 km/s and 1.5 km/s respectively.

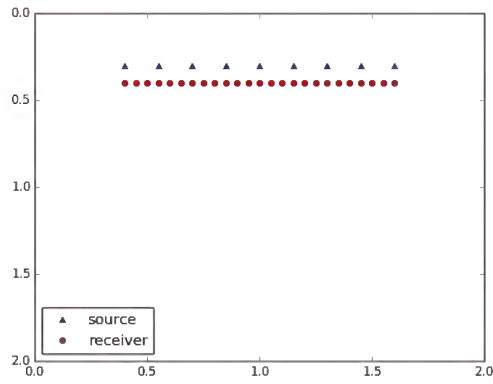


Figure 3.10: Multi-layer example: sources and receivers, where sources and receivers are located on the top of the domain.

bottom structures. Figure 3.11 (a) is a plot of the image using linear color scale, where top reflector can be easily observed; figure 3.11 (b) uses log color scale to reveal the bottom reflectors.

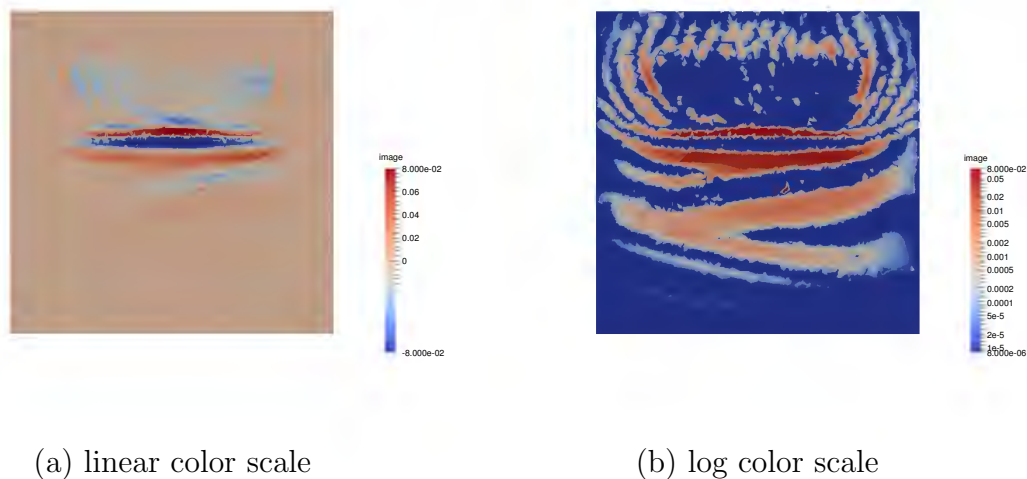


Figure 3.11: Results of the multi-layer RTM example: (a) plot of the image using a linear color scale; (b) plot of the image using a log color scale.

3.5.4 Marmousi model

The last test case in this chapter is the Marmousi model. Since the Marmousi model is a 2D model but our solver is implemented for 3D problems, we extend it to a 2.5D model. Here, we extrude the model along y -axis and let the model be constant along y -dimension.

The model is sampled to a $36 \text{ m} \times 36 \text{ m} \times 36 \text{ m}$ hexahedral grid. The grid size is $256 \times 9 \times 81$, which sets the model size to $9216 \text{ m} \times 324 \text{ m} \times 2916 \text{ m}$. The model above $z = 252 \text{ m}$ (first 7 cells in z -direction) is a water layer with a constant velocity of 1500 m/s . The velocity range in the model is $[1500 \text{ m/s}, 5500 \text{ m/s}]$. Figure 3.12 and 3.13 plots the Marmousi model in both 2D and 3D views.

The sources, which are the Ricker pulse of 15 Hz peak frequency, are placed along the line $z = 150 \text{ m}$. 50 shots are deployed evenly starting from $x = 100 \text{ m}$ to $x = 9000 \text{ m}$. There are 249 receivers equally spaced every 36 m from $x = 126 \text{ m}$ to $x = 9054 \text{ m}$ and $z = 200 \text{ m}$. The configuration of the sources and receivers is illustrated in

figure 3.14. The total time of the wave simulation is $T = 5.0$ s. The domain is built on a structured hexahedral mesh, and the wave equations are solved by GL-HEX algorithm.

Figure 3.15 and 3.16 report the resulting RTM image plotted in linear and log color scales respectively. Subsurface structures such as layers and faults can be observed in the RTM image.

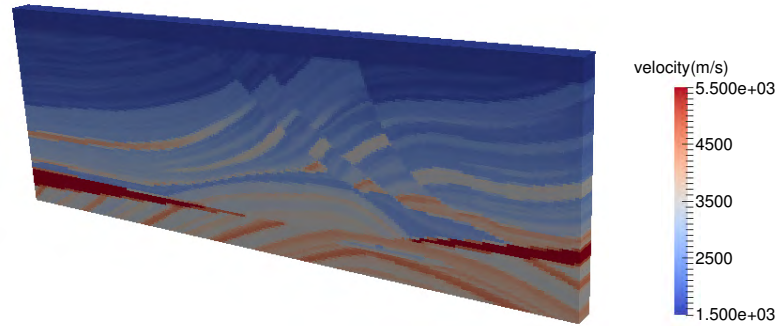


Figure 3.12: Marmousi velocity model (viewed in 3D). Velocities are in the range of 1500 m/s to 5500 m/s.

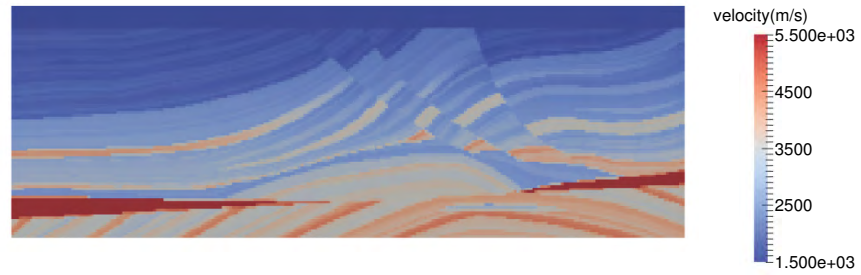


Figure 3.13: Marmousi velocity model (viewed in 2D). Velocities are in the range of 1500 m/s to 5500 m/s.

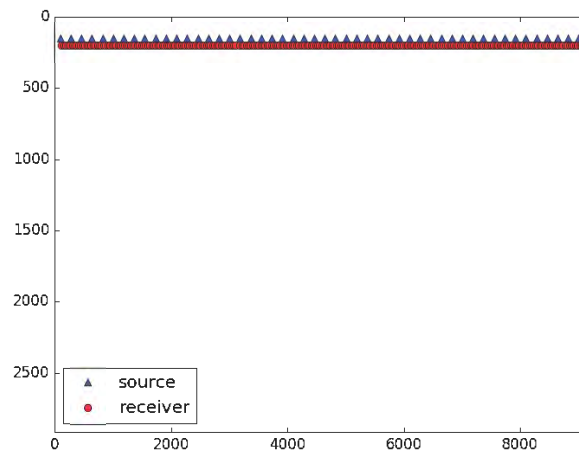


Figure 3.14: Marmousi example: sources and receivers are placed at $z = 150$ m and $z = 200$ m respectively.

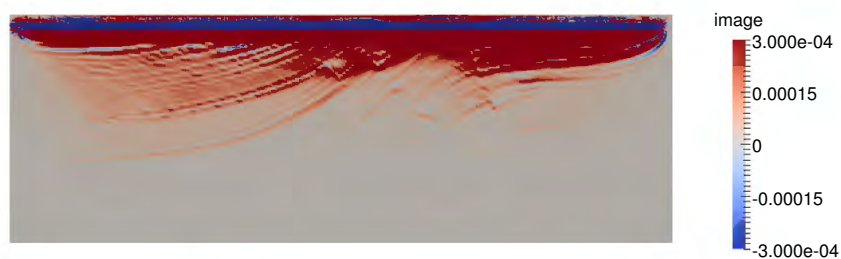


Figure 3.15: Marmousi example: RTM image of Marmousi (plotted in a linear color scale).

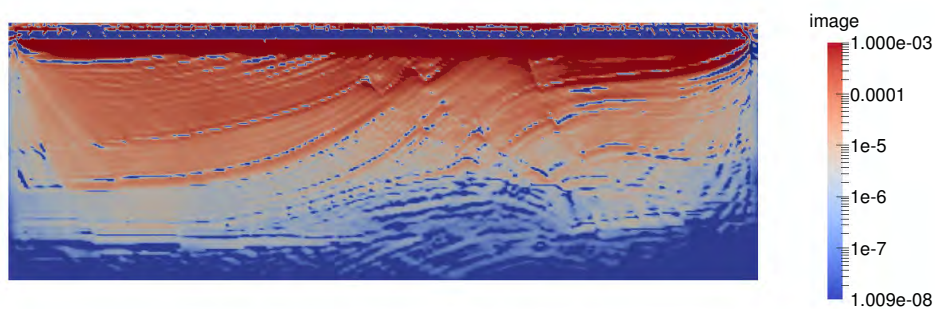


Figure 3.16: Marmousi example: RTM image of Marmousi (plotted in a log color scale).

3.6 Summary

This chapter studies the application of DG methods in reverse time migration (RTM). RTM utilizes two-way wavefields to evaluate an imaging condition through cross-correlation, and the output image reveals the subsurface structures.

To study the wave simulation in the context of seismic imaging, this chapter first introduces preliminaries of perfectly matched layers (PML) and point source injection. PML absorbs wave energy near the boundary of the domain, and allows us to truncate a domain into a region of interest. We can thereby study wave problems posed on unbounded domains, which is typically the case in seismic imaging. In seismic surveys, the source signal generated by airguns and explosives may be modeled as point sources. To inject such sources into the wave system, we implement the source injection using a scattered-total field formulation to avoid the singularities introduced by the point source.

RTM evaluates the cross-correlation of source and receiver wavefields at the same physical time. This process consumes a large amount of memory in most classical RTM implementations. To avoid such data movement, we adopt the approach in [9]: we save the boundary values of the forward wavefield at each time step in memory, and provide these values as boundary conditions to recover the forward wavefield at the backward stage. With an additional cost of floating point operations, the memory cost is significantly reduced.

To improve the imaging condition, we replace the pressure fields in the imaging condition with the characteristic fields. The improvement of image quality by using characteristic fields is reported in [9]. We also employ high-order quadrature rules to achieve high-order accuracy in the time integration of the imaging condition. This technique is based on the polynomial representation of Adam-Bashforth evolution.

At the end of this chapter, we validate the DG-RTM implementation with three

test cases—a single layer model, a multi-layer model and the Marmousi model. In the RTM images generated by our implementation, we observe expected subsurface structures such as layers and faults.

In the next chapter, we will further extend DG-RTM to DG-FWI which is a more sophisticated seismic algorithm.

Full waveform inversion

Building from DG-RTM, I extend the work to full waveform inversion (FWI) in this chapter. Different from RTM which recovers subsurface structures without realizing their physical properties, FWI solves an inverse problem and recovers subsurface physical parameters.

FWI was first pioneered in the 1980s [32, 33] and its study of 2D synthetic surveys on acoustic waves was soon carried out [126]. As the development of computer technology, nowadays FWI can solve 3D elastic problems [127] and has been applied to real data [128].

Formulated as a PDE-constraint optimization problem, FWI seeks the minimum mismatch between synthetic and recorded data in terms of both traveltimes and waveform. However, this problem is ill-posed, and the solution is not guaranteed to be unique. The optimization path can be trapped in local minima, which is known as the cycle-skipping issue [6]. The importance of low-frequency data is realized to design well-posed FWI [129], but the acquisition of low-frequency data is challenging. Alternatives to design better-posed FWI are: adding regularization terms to guide the optimization paths [34, 39], using different objective functions [40], and applying frequency continuation technique [41]. Additionally, to avoid the cycle-skipping issue,

FWI must start with good initial guesses, which can be provided by other seismic algorithms such as traveltime tomography [3].

FWI is computationally intensive, especially for 3D cases. To address large-scale problems, FWI is typically implemented in parallel. Intuitive parallelization approaches are distributing each shot to a processor and solving wave equations sequentially on each processor [130]. More sophisticated ways to implement FWI is to introduce domain decomposition and assign each subdomain to a processor [131], which is also adopted in our work. Most of these parallel implementations are realized on CPUs, while the study of GPU-accelerated FWI has been carried out in recent years [132]. Supplementary to parallel computing, many other techniques such as preconditioners [133] and source encoding [134] are developed to save the computational expenses.

Similar to RTM, FWI requires solving wave equations multiple times. To solve these wave equations, FWI has been studied on various numerical solvers [6, 10], most of which are finite difference methods [13, 14, 15, 16]. As to my knowledge, only a few papers have been published for complete DG-FWI procedures. Wilcox et al. [38] discussed how to derive discretely exact derivatives in the context of DG-based PDE-constraint optimization. Ober et al. [28] reported their results of DG-FWI for visco-TTI elastic problems. However, DG-FWI with multi-rate time stepping and multi-GPU acceleration has rarely been discussed in the literature.

In this chapter, I first study the adjoint-state method to obtain the gradient of the objective function in FWI. Afterward, I employ the steepest descent method to solve the optimization problem and validate the conventional FWI algorithm with test cases. To highlight the advantage of DG methods, I focus on inverting sharp interfaces at the end of this chapter, where we specify certain perturbations at media interfaces. The impact of the interface perturbation is given by integrating the RTM image over

element surfaces. In addition, meshes are regenerated in the FWI iterations to align with the media interfaces.

4.1 Formulation

Recall that the first order acoustic wave equation is given by

$$\frac{\partial p(\mathbf{x}, t)}{\partial t} + \rho c^2 \nabla \cdot \mathbf{v}(\mathbf{x}, t) = f(\mathbf{x}, t), \quad \mathbf{x} \in \Omega, \quad (4.1a)$$

$$\frac{\partial \mathbf{v}(\mathbf{x}, t)}{\partial t} + \frac{1}{\rho} \nabla p(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \Omega, \quad (4.1b)$$

$$p(\mathbf{x}, 0) = 0, \quad \mathbf{x} \in \Omega, \quad (4.1c)$$

$$\mathbf{v}(\mathbf{x}, 0) = \mathbf{0}, \quad \mathbf{x} \in \Omega, \quad (4.1d)$$

where f is the source signal, c is the velocity of the media and ρ is the density of the media, and a proper boundary condition (such as absorbing boundary) is posed at the boundary of the domain.

As stated in chapter 1, the velocity c and density ρ are in the model space, and can be denoted by $\mathbf{m}(c, \rho)$. The wave equation is an operator acting on the model parameter to obtain the solution $\mathbf{u}(\mathbf{x}, t) = (p(\mathbf{x}, t), \mathbf{v}(\mathbf{x}, t))$, which is denoted by $\mathcal{F}[\mathbf{m}] = \mathbf{u}$. We can use a numerical solver (e.g. our DG solver) to solve the wave equation with a given model, but it is challenging and interesting to inversely solve the model parameters \mathbf{m} . To this end, we formulate the inversion as an optimization problem, and begin the introduction by discussing more seismic parameters.

In a seismic survey, we use many receivers to record the wave signals as seismic data. Denote the data recorded at receivers as $\mathbf{d}_{r,s}(t) = \mathbf{u}_s(\mathbf{x}_r, t)$ where s and r are the indexes of the shots and receivers respectively, and $\mathbf{d}(t)$ as a collection of $\mathbf{d}_{r,s}(t)$.

Define operator S that maps the wavefield to the traces at receiver locations as

$$S\mathbf{u}(\mathbf{x}, t) = \mathbf{u}(\mathbf{x}_r, t), \quad (4.2)$$

where \mathbf{x}_r is the location of the receiver. We then have

$$\begin{aligned} \mathcal{J}[\mathbf{m}] &= \frac{1}{2} \langle S\mathcal{F}[\mathbf{m}] - \mathbf{d}, S\mathcal{F}[\mathbf{m}] - \mathbf{d} \rangle \\ &= \frac{1}{2} \|S\mathcal{F}[\mathbf{m}] - \mathbf{d}\|_2^2, \end{aligned} \quad (4.3)$$

where $\|\mathbf{d}\|_2^2 = \sum_s \int_0^T \int_\Omega \sum_r (\delta(\mathbf{x}_r) d_{r,s})^2 d\mathbf{x} dt = \sum_{r,s} \int_0^T (d_{r,s})^2 dt$. The model parameter \mathbf{m} is then solved by

$$\mathbf{m} = \arg \min \mathcal{J}[\mathbf{m}]. \quad (4.4)$$

In other words, we need to solve the optimization problem

$$\min_{\mathbf{m}} \mathcal{J}[\mathbf{m}]. \quad (4.5)$$

FWI is challenging because the problem is ill-posed and there is no guarantee of finding the global minimum [6]. Typically, FWI requires a close enough initial guess for the optimization to attain the global minimum [6].

To apply a gradient-based optimization scheme to problem (4.5), one needs to compute the derivatives of \mathcal{J} with respect to \mathbf{m} . The intuitive approach, in which one approximates the derivative by its definition in a finite difference manner, is unaffordable for modern computers. Therefore, we introduce the adjoint-state method to compute the derivatives in the following two sections.

4.2 Continuous adjoint-state method for FWI

In this section and next section, we discuss the adjoint-state method to obtain the gradient of the misfit function \mathcal{J} . The two sections have different highlights: this section focuses on the derivation at continuous level; the next section studies the discrete adjoint-state method and is based on the discretization of the DG formulation.

A variety of derivations is available for the continuous adjoint-state method (e.g. Gauthier et al. [126]). In this section, we mostly follow Laurent Demanet's notes on Waves and Imaging [135] to introduce the adjoint-state method. For convenience, we use the second order wave equation

$$\frac{1}{c^2} \frac{\partial^2 u(\mathbf{x}, t)}{\partial t^2} - \Delta u(\mathbf{x}, t) = f(\mathbf{x}, t), \quad \mathbf{x} \in \Omega \quad (4.6)$$

where c is the same velocity as in (4.1), u is the wave solution and f is the source pulse. The derivation of the adjoint-state method for first-order wave equation (4.1) can be obtained analogously as it is convertible between the first order wave equation and the second order wave equation.

4.2.1 Born approximation

We first study the impact of the perturbation in the velocity model. A small perturbation in velocity leads to a perturbation in the wavefield. The linearization of the wavefield perturbation is known as the Born approximation, which is analogous to the first order term in Taylor expansion of a general function [136].

For simplicity, denote the model parameter $m = 1/c^2$, and equation (4.6) becomes

$$m \frac{\partial^2 u}{\partial t^2} - \Delta u = f, \quad (4.7)$$

where m is now linearly acting on the wave equation.

To study the impact of the variation of m on the wave system, we let m be interpreted as an combination of an initial model m_0 and a small perturbation δm ,

$$m = m_0 + \delta m. \quad (4.8)$$

As a result, the wave-field u is also split as

$$u = u_0 + \delta u, \quad (4.9)$$

where u_0 is called the incident field, δu is known as the scattered field, and u and u_0 solve the wave equation with model parameters m and m_0 respectively

$$m \frac{\partial^2 u}{\partial t^2} - \Delta u = f, \quad (4.10)$$

and

$$m_0 \frac{\partial^2 u_0}{\partial t^2} - \Delta u_0 = f. \quad (4.11)$$

Subtracting (4.11) from (4.10) and utilizing the linearity of the wave equation, we arrive at,

$$m_0 \frac{\partial^2 \delta u}{\partial t^2} + \delta m \frac{\partial^2 u_0}{\partial t^2} + \delta m \frac{\partial^2 \delta u}{\partial t^2} - \Delta \delta u = 0 \quad (4.12)$$

Dropping the second order small term in the above equation , we obtain

$$m_0 \frac{\partial^2 \delta u}{\partial t^2} - \Delta \delta u \approx -\delta m \frac{\partial^2 u_0}{\partial t^2}. \quad (4.13)$$

Therefore, the solution δu to the equation

$$\left(m_0 \frac{\partial^2}{\partial t^2} - \Delta\right) \delta u = -\Delta m \frac{\partial^2 u_0}{\partial t^2}, \quad (4.14a)$$

$$\left(m_0 \frac{\partial^2}{\partial t^2} - \Delta\right) u_0 = f, \quad (4.14b)$$

is known as the perturbation fields, and we denote $F[m_0]\delta m = \delta u$.

Using notation \mathcal{F} and its linearization F , we have $\mathcal{F}[m + \delta m] = \mathcal{F}[m] + F[m]\delta m + \mathcal{O}(\|\delta m\|^2)$, which is known as the Born expansion [135, 136]. Substituting the forward operator in the misfit expression (4.3) yields

$$\begin{aligned} \mathcal{J}[m + \delta m] &= \frac{1}{2} \langle S\mathcal{F}[m + \delta m] - \mathbf{d}, S\mathcal{F}[m + \delta m] - \mathbf{d} \rangle \\ &= \frac{1}{2} \langle S\mathcal{F}[m] - \mathbf{d}, S\mathcal{F}[m] - \mathbf{d} \rangle + \langle SF[m]\delta m, S\mathcal{F}[m] - \mathbf{d} \rangle + \mathcal{O}(\|\delta m\|^2) \\ &= \mathcal{J}[m] + \langle SF[m]\delta m, S\mathcal{F}[m] - \mathbf{d} \rangle + \mathcal{O}(\|\delta m\|^2), \\ &= \mathcal{J}[m] + \langle \delta m, F^*S^*(S\mathcal{F}[m] - \mathbf{d}) \rangle + \mathcal{O}(\|\delta m\|^2), \end{aligned} \quad (4.15)$$

where F^* is the adjoint of F and S^* is the adjoint of S . This gives the Fréchet derivative of \mathcal{J} as

$$D\mathcal{J}[m] = F^*S^*(S\mathcal{F}[m] - \mathbf{d}). \quad (4.16)$$

With expression (4.16), the computation of the Fréchet derivative of \mathcal{J} becomes a problem of obtaining the adjoint operator F^*S^* , which will be discussed next.

4.2.2 Imaging condition

In equation (4.15) and (4.16), we have introduced the adjoint operator F^* and S^* without clearly defining it. This section discusses the derivation of F^*S^* and gives its explicit representation which is also known as the imaging condition [135].

The adjoint operator S^* is defined by

$$\langle d_r, Su \rangle = \langle S^* d_r, u \rangle, \quad (4.17)$$

where u is the solution defined on domain Ω and d_r is the traces defined on receiver locations. S maps wavefield to given receiver locations (defined in (4.2)). We then have

$$\int_0^T (Su)(t) d_r(t) dt = \int_0^T \int_{\Omega} d_r(t) u(\mathbf{x}, t) \delta(\mathbf{x} - \mathbf{x}_r) dx dt. \quad (4.18)$$

Therefore, the adjoint operator S^* is defined by $S^* d_r = d_r(t) \delta(\mathbf{x} - \mathbf{x}_r)$.

To get an explicit definition of F^* , we look at the definition of the adjoint operator F^* given by

$$\langle d, Fm \rangle = \langle F^* d, m \rangle, \quad (4.19)$$

where $d(\mathbf{x}, t) = \sum_r d_r(t) \delta(\mathbf{x} - \mathbf{x}_r)$ is the data collected at the receivers and m is a model perturbation. In fact, in the following derivation, d and m do not necessarily need a physical interpretation and can be arbitrary functions of (\mathbf{x}, t) .

Since the left-hand side of (4.19) is well defined, we have

$$\langle d, Fm \rangle = \int_0^T \int_{\Omega} d(\mathbf{x}, t) u(\mathbf{x}, t) d\mathbf{x} dt, \quad (4.20)$$

where $u = F[m_0]m$, as defined in (4.14), is the first order perturbation of the incident field, and is given by

$$\left(m_0 \frac{\partial^2}{\partial t^2} - \Delta \right) u = -m \frac{\partial^2 u_0}{\partial t^2}, \quad (4.21a)$$

$$\left(m_0 \frac{\partial^2}{\partial t^2} - \Delta \right) u_0 = f. \quad (4.21b)$$

In order to take the definition of u into inner product (4.20), we consider the

following wave equation with $d(\mathbf{x}, t)$ as a right-hand side,

$$\left(m_0 \frac{\partial^2}{\partial t^2} - \Delta\right) \lambda(\mathbf{x}, t) = d(\mathbf{x}, t). \quad (4.22)$$

Substituting (4.22) into (4.20), and integrating by parts in both time and space, we have

$$\begin{aligned} \langle d, Fm \rangle &= \int_0^T \int_{\Omega} \lambda(\mathbf{x}, t) \left(m_0 \frac{\partial^2}{\partial t^2} - \Delta\right) u(\mathbf{x}, t) d\mathbf{x} dt \\ &\quad + \int_{\Omega} m_0 \frac{\partial \lambda}{\partial t} u|_0^T d\mathbf{x} - \int_{\Omega} m_0 \lambda \frac{\partial u}{\partial t}|_0^T d\mathbf{x} \\ &\quad + \int_0^T \int_{\partial\Omega} \frac{\partial \lambda}{\partial n} u d\mathbf{x} dt - \int_0^T \int_{\partial\Omega} \lambda \frac{\partial u}{\partial n} d\mathbf{x} dt, \end{aligned} \quad (4.23)$$

where Ω is the domain of our interest. Assuming we have boundary condition

$$u|_{\partial\Omega} = 0, \quad (4.24a)$$

$$\lambda|_{\partial\Omega} = 0, \quad (4.24b)$$

the last two terms vanish in (4.23). The other boundary terms vanish when the following conditions are satisfied,

$$u|_{t=0} = \frac{\partial u}{\partial t}|_{t=0} = 0, \quad (4.25a)$$

$$\lambda|_{t=T} = \frac{\partial \lambda}{\partial t}|_{t=T} = 0, \quad (4.25b)$$

Condition (4.25a) is the initial conditions for u . In contrast, condition (4.25b) imposes terminal conditions instead of initial conditions on q , which requires us to solve the field q backwardly. Assuming conditions (4.25) are satisfied, we then return to

(4.23) and simplify it to have

$$\begin{aligned}\langle d, Fm \rangle &= \int_0^T \int_{\mathbb{R}^3} \lambda(\mathbf{x}, t) \left(m_0 \frac{\partial^2}{\partial t^2} - \Delta \right) u(\mathbf{x}, t) d\mathbf{x} dt \\ &= - \int_{\mathbb{R}^3} \left(\int_0^T \lambda(\mathbf{x}, t) \frac{\partial^2 u_0}{\partial t^2} dt \right) m(\mathbf{x}) d\mathbf{x}.\end{aligned}\tag{4.26}$$

According to the definition of F^* , we should have

$$\langle F^* d, m \rangle = - \int_{\mathbb{R}^3} \left(\int_0^T \lambda(\mathbf{x}, t) \frac{\partial^2 u_0}{\partial t^2} dt \right) m(\mathbf{x}) d\mathbf{x},\tag{4.27}$$

which forces

$$F^* d = - \int_0^T \lambda(\mathbf{x}, t) \frac{\partial^2 u_0}{\partial t^2} dt.\tag{4.28}$$

This equation is called the imaging condition, and it is analogous to the imaging condition (3.16) in the reverse time migration discussed in chapter 3.

When dealing with multi-sources, we can extend equation (4.28) to

$$F^* d = - \sum_s \int_0^T \lambda_s(\mathbf{x}, t) \frac{\partial^2 u_{s,0}}{\partial t^2} dt.\tag{4.29}$$

where s is the index of the source.

Now, we are able to obtain the first order derivative of the objective function \mathcal{J} using equation (4.16). One can easily get the Fréchet derivative of \mathcal{J} with respect to velocity c using chain rule, and obtain

$$\frac{\partial \mathcal{J}}{\partial c} = \frac{\partial \mathcal{J}}{\partial m} \frac{\partial m}{\partial c} = \frac{-2}{c^3} \frac{\partial \mathcal{J}}{\partial m}\tag{4.30}$$

Equation (4.22) is called the adjoint equation. Analogously, we can write the the

adjoint equation for the first order wave system (4.1) as

$$\frac{\partial p}{\partial t} + \nabla \cdot \left(\frac{1}{\rho} \mathbf{v} \right) = d, \quad (4.31a)$$

$$\frac{\partial \mathbf{v}}{\partial t} + \nabla (\rho c^2 p) = 0, \quad (4.31b)$$

and it is derived using integration by part as we did in (4.23).

The final result we have from this derivation is a Fréchet derivative, and is defined continuously everywhere in the domain Ω . This Fréchet derivative is not the gradient in common sense. In particular, it is an linear operator which maps the perturbation of the model parameter to the perturbation of the misfit function. As stated in equation (4.15)

$$\begin{aligned} D\mathcal{J}[m][\delta m] &= \mathcal{J}[m + \delta m] - \mathcal{J}[m] \\ &= \langle F^* S^* (S\mathcal{F}[m] - \mathbf{d}), \delta m \rangle \\ &= \int_{\Omega} (F^* S^* (S\mathcal{F}[m] - \mathbf{d})) \delta m \, d\mathbf{x}. \end{aligned} \quad (4.32)$$

However, in the DG method, the model parameters are represented as piece-wise constants over elements, and hence we need to know the “real” gradient of the misfit function with respect to the discrete model parameters. An intuitive approach is taking an average of the imaging condition over each element. This approach, known as optimize-then-discretize, suffers from numerical errors from the discretizations. In contrast, the discrete adjoint-state method takes the PDE discretization into consideration and the corresponding gradient is free of numerical errors. In the next section, we will discuss the discrete adjoint-state method in the context of the DG methods.

4.3 Discrete adjoint-state method for DG-FWI

When we apply the adjoint-state method to a certain numerical scheme, there are two approaches. The first one, known as optimize-then-discretize, derives the gradient at the continuous level and then discretizes the continuous gradient regardless of the discretization of the PDEs. This approach is highly affected by the numerical error of the PDE solver. As the numerical error of the PDE solver increases, the approximation of the gradient deteriorates. The second approach, known as discretize-then-optimize, applies the adjoint-state method to the discretized PDE system, and the gradient is also given in a discrete form which already takes the PDE discretization into consideration. This approach may lead to “discretely exact derivative” which is exact up to machine precision [38].

In 2015, Wilcox et al. [38] gave the formulation of the gradient for a variety of PDE-constrained optimization problems in the context of discontinuous Galerkin discretization. As the result of the paper, from forward discretization to adjoint discretization, the weak(strong) form formulation becomes strong(weak) form formulation, and the upwind numerical flux becomes downwind numerical flux. In this section, we follow the standard framework of discrete adjoint-state method [137], generalize the result of Wilcox’s paper, and present a detailed derivation based on the acoustic wave equations.

4.3.1 The derivation

The DG semi-discrete formulation of the forward problem (4.1) is given by,

$$\begin{aligned} M\dot{\mathbf{u}} + A\mathbf{u} &= \mathbf{f}, \\ \mathbf{u}(0) &= 0, \end{aligned} \tag{4.33}$$

where M is the mass matrix, A is the right-hand side contribution depending on model $\mathbf{m}(c, \rho)$, and $\mathbf{u} = (p, \mathbf{v})$ is the numerical solution to the problem.

The fully discrete form with 3-step Adams-Bashforth scheme is

$$M \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \sum_{i=0}^2 \alpha_i A \mathbf{u}^{n-i} = \mathbf{f}^n, \quad (4.34)$$

with initial condition

$$\mathbf{u}^0 = \mathbf{u}^{-1} = \mathbf{u}^{-2} = 0, \quad (4.35)$$

and AB3 coefficients

$$\alpha_0 = \frac{23}{12}, \quad \alpha_1 = -\frac{16}{12}, \quad \alpha_2 = \frac{5}{12}. \quad (4.36)$$

Let N_q denote the total degrees of freedom in the DG discretization, then M and A are of size $N_q \times N_q$; \mathbf{u}^n and \mathbf{f}^n are of size $N_q \times 1$. Let N_r be the number of receivers. $\mathbf{d}_{N_r \times 1}$ is the data at the receiver locations and $V_{N_r \times N_q}$ is a Vandermonde matrix that restrict the DG solution at the receiver locations,

$$V = \begin{bmatrix} \varphi_1(\mathbf{x}_1) & \varphi_2(\mathbf{x}_1) & \dots & \varphi_{N_q/4}(\mathbf{x}_1) & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ \varphi_1(\mathbf{x}_{N_r}) & \varphi_2(\mathbf{x}_{N_r}) & \dots & \varphi_{N_q/4}(\mathbf{x}_{N_r}) & 0 & \dots & 0 \end{bmatrix}, \quad (4.37)$$

where φ_i are the basis functions, and $\varphi_1, \dots, \varphi_{N_q/4}$ are the basis functions corresponding to the pressure field (1/4 of the total degrees of freedom in our formulation). Because we only record the data in the pressure field, the right part of the Vandermonde matrix V are all zeros.

Using the above notation, the misfit function can be written in a semi-discrete

form as

$$\mathcal{J} = \frac{1}{2} \int_0^T (V\mathbf{u} - \mathbf{d})^T (V\mathbf{u} - \mathbf{d}) dt, \quad (4.38)$$

and its fully discrete form is

$$\mathcal{J} = \frac{\Delta t}{2} \sum_{n=0}^N (V\mathbf{u}^n - \mathbf{d}^n)^T (V\mathbf{u}^n - \mathbf{d}^n). \quad (4.39)$$

Define

$$\mathbf{h}^n := M \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \sum_{i=0}^2 \alpha_i A \mathbf{u}^{n-i} - \mathbf{f}^n, \quad (4.40)$$

$$j^n := \frac{1}{2} (V\mathbf{u}^n - \mathbf{d}^n)^T (V\mathbf{u}^n - \mathbf{d}^n). \quad (4.41)$$

The discrete optimization problem is stated as

$$\min_{c, \rho} \mathcal{J} := \Delta t \sum_{n=0}^N j^n, \quad (4.42)$$

subject to $\mathbf{h}^n = 0$ and $\mathbf{u}^0 = \mathbf{u}^{-1} = \mathbf{u}^{-2} = 0$, $\forall n$.

We now introduce Langrange multiplier and use it for the following derivation. Denote vectors $\boldsymbol{\lambda}_{N_q \times 1}^n$, $\boldsymbol{\mu}_{N_q \times 1}^{-2}$, $\boldsymbol{\mu}_{N_q \times 1}^{-1}$, and $\boldsymbol{\mu}_{N_q \times 1}^{-0}$ as the Lagrange multiplier. We have the Lagrangian

$$\mathcal{L} := \Delta t \sum_{n=0}^N (j^n + (\boldsymbol{\lambda}^n)^T \mathbf{h}^n) + \sum_{i=-2}^0 (\boldsymbol{\mu}^i)^T \mathbf{u}^i, \quad (4.43)$$

Since $\mathbf{h}^n \equiv 0$ and $\mathbf{u}^i \equiv 0$, we are free to choose $\boldsymbol{\lambda}^n$ and $\boldsymbol{\mu}^i$.

We take derivative of \mathcal{L} with respect to each entry of $\mathbf{m}(c, \rho)$, and obtain

$$\frac{d\mathcal{L}}{d\mathbf{m}} = \Delta t \sum_{n=0}^N \left[\left(\frac{\partial j^n}{\partial \mathbf{u}^n} \right)^T \frac{d\mathbf{u}^n}{d\mathbf{m}} + (\boldsymbol{\lambda}^n)^T \left(\frac{d\mathbf{h}^n}{d\mathbf{m}} + \sum_{i=-2}^1 \frac{\partial \mathbf{h}^n}{\partial \mathbf{u}^{n+i}} \frac{d\mathbf{u}^{n+i}}{d\mathbf{m}} \right) \right] \quad (4.44)$$

Consider the last term in summation (4.44)

$$\begin{aligned} & \Delta t \sum_{n=0}^N \left[(\boldsymbol{\lambda}^n)^T \sum_{i=-2}^1 \frac{\partial \mathbf{h}^n}{\partial \mathbf{u}^{n+i}} \frac{d\mathbf{u}^{n+i}}{d\mathbf{m}} \right] \\ &= \Delta t \sum_{n=0}^N \left[(\boldsymbol{\lambda}^n)^T \left(\frac{M}{\Delta t} \frac{d(\mathbf{u}^{n+1} - \mathbf{u}^n)}{d\mathbf{m}} + \sum_{i=0}^2 \alpha_i A \frac{d\mathbf{u}^{n-i}}{d\mathbf{m}} \right) \right] \end{aligned} \quad (4.45)$$

Assume terminal condition on $\boldsymbol{\lambda}$ as

$$\boldsymbol{\lambda}^N = \boldsymbol{\lambda}^{N+1} = \boldsymbol{\lambda}^{N+2} = 0 \quad (4.46)$$

By changing the order of summation, equation (4.45) can be simplified into

$$\Delta t \sum_{n=0}^N \left[\left(\frac{d\mathbf{u}^n}{d\mathbf{m}} \right)^T \left(M \frac{\boldsymbol{\lambda}^{n-1} - \boldsymbol{\lambda}^n}{\Delta t} + \sum_{i=0}^2 \alpha_i A^T \boldsymbol{\lambda}^{n+i} \right) \right]. \quad (4.47)$$

Substitute (4.47) back into (4.44), we have,

$$\frac{d\mathcal{L}}{d\mathbf{m}} = \Delta t \sum_{n=0}^N \left[\left(\frac{d\mathbf{u}^n}{d\mathbf{m}} \right)^T \left(M \frac{\boldsymbol{\lambda}^{n-1} - \boldsymbol{\lambda}^n}{\Delta t} + \sum_{i=0}^2 \alpha_i A^T \boldsymbol{\lambda}^{n+i} + V^T (V\mathbf{u}^n - \mathbf{d}^n) \right) + (\boldsymbol{\lambda}^n)^T \frac{d\mathbf{h}^n}{d\mathbf{m}} \right]. \quad (4.48)$$

We want to avoid the evaluation of $d\mathbf{u}^n/d\mathbf{m}$, so we let

$$M \frac{\boldsymbol{\lambda}^{n-1} - \boldsymbol{\lambda}^n}{\Delta t} + \sum_{i=0}^2 \alpha_i A^T \boldsymbol{\lambda}^{n+i} + V^T (V\mathbf{u}^n - \mathbf{d}^n) = 0, \quad (4.49)$$

which is known as the adjoint equation.

Since $\mathbf{h}^n \equiv 0$ and $\mathbf{u}^0 = \mathbf{u}^{-1} = \mathbf{u}^{-2} \equiv 0$, considering the relation $d\mathcal{J}/d\mathbf{m} = d\mathcal{L}/d\mathbf{m}$,

we have the gradient of the misfit function.

$$\begin{aligned}\frac{d\mathcal{J}}{d\mathbf{m}} &= \frac{d\mathcal{L}}{d\mathbf{m}} \\ &= \Delta t \sum_{n=0}^N (\boldsymbol{\lambda}^n)^T \frac{d\mathbf{h}^n}{d\mathbf{m}},\end{aligned}\tag{4.50}$$

where $\boldsymbol{\lambda}^n$ are given by the adjoint equation (4.49) and \mathbf{h}^n are defined in (4.40).

In the following subsections, we will discuss how to obtain $\boldsymbol{\lambda}^n$ and $d\mathbf{h}^n/d\mathbf{m}$ in detail.

4.3.2 The adjoint equation

Equation (4.49) is the adjoint equation, which has zero terminal condition and should be solved backward in time. Compared to the forward problem (4.34), this equation transforms the forward Adams-Bashforth to the backward Adams-Bashforth, and matrix A to A^T . To better understand A^T , we decompose A as

$$A = C_1 D + C_2 S,\tag{4.51}$$

where

$$C_1 = \begin{bmatrix} \rho_1 c_1^2 & & & & \\ & \ddots & & & \\ & & \rho_K c_K^2 & & \\ & & & \rho_1^{-1} & \\ & & & & \ddots \\ & & & & & \rho_K^{-1} \end{bmatrix}\tag{4.52}$$

and

$$C_2 = \begin{bmatrix} c_1 & & & & \\ & \ddots & & & \\ & & c_K & & \\ & & & c_1 & \\ & & & & \ddots \\ & & & & & c_K \end{bmatrix} \quad (4.53)$$

are coefficient matrices, and c_k, ρ_k are the constant velocity and density in element k .

Matrix D is the combination of differentiation operator and central numerical fluxes, which contains the following terms,

- Strong form

$$(\nabla \cdot \mathbf{v}, \varphi)_{D^k} + \left(\frac{1}{2} \mathbf{n} \cdot \llbracket \mathbf{v} \rrbracket, \varphi \right)_{\partial D^k}, \quad (4.54a)$$

$$\left(\frac{\partial p}{\partial x_1}, \varphi \right)_{D^k} + \left(\frac{1}{2} n_1 \llbracket p \rrbracket, \varphi \right)_{\partial D^k}, \quad (4.54b)$$

$$\left(\frac{\partial p}{\partial x_2}, \varphi \right)_{D^k} + \left(\frac{1}{2} n_2 \llbracket p \rrbracket, \varphi \right)_{\partial D^k}, \quad (4.54c)$$

$$\left(\frac{\partial p}{\partial x_3}, \varphi \right)_{D^k} + \left(\frac{1}{2} n_3 \llbracket p \rrbracket, \varphi \right)_{\partial D^k}. \quad (4.54d)$$

- Weak form

$$-(\mathbf{v}, \nabla \varphi)_{D^k} + \left(\frac{1}{2} \mathbf{n} \cdot \{\{\mathbf{v}\}\}, \varphi \right)_{\partial D^k}, \quad (4.55a)$$

$$-\left(p, \frac{\partial \varphi}{\partial x_1} \right)_{D^k} + \left(\frac{1}{2} n_1 \{\{p\}\}, \varphi \right)_{\partial D^k}, \quad (4.55b)$$

$$-\left(p, \frac{\partial \varphi}{\partial x_2} \right)_{D^k} + \left(\frac{1}{2} n_2 \{\{p\}\}, \varphi \right)_{\partial D^k}, \quad (4.55c)$$

$$-\left(p, \frac{\partial \varphi}{\partial x_3} \right)_{D^k} + \left(\frac{1}{2} n_3 \{\{p\}\}, \varphi \right)_{\partial D^k}. \quad (4.55d)$$

Here the test function φ is selected over the test space, and D^k is the k th element. For the boundary condition, we assume either Dirichlet boundary condition ($p^+ = -p^-$, $\mathbf{v}^+ = \mathbf{v}^-$) or first order absorbing boundary condition ($p^+ = 0$, $\mathbf{v}^+ = 0$). In addition, we also assume exact volume and surface integration/quadrature. With these assumptions, matrix D has the skew-symmetric property,

$$D = -D^T. \quad (4.56)$$

If we assume an upwind flux is used in the DG formulation, matrix S represents the penalty term that stabilizes the jumps in pressure p and the jumps in the normal direction of velocity \mathbf{v} , and it is contributed by the following terms

$$- (\llbracket p \rrbracket, \varphi)_{\partial D^k}, \quad (4.57a)$$

$$- (n_1 \llbracket \mathbf{v} \rrbracket, \varphi)_{\partial D^k}, \quad (4.57b)$$

$$- (n_2 \llbracket \mathbf{v} \rrbracket, \varphi)_{\partial D^k}, \quad (4.57c)$$

$$- (n_3 \llbracket \mathbf{v} \rrbracket, \varphi)_{\partial D^k}, \quad (4.57d)$$

where φ are the basis functions. Again, with the assumption of exact quadrature, S is a symmetric matrix,

$$S = S^T. \quad (4.58)$$

With the property (4.56) and (4.58), we can easily obtain

$$A^T = -DC_1 + SC_2. \quad (4.59)$$

Noticing that the above result requires exact quadrature. In the case where the exact quadrature is not available, the results in Wilcox's paper [38] can be applied, where the strong and weak form formulations are swapped, and the upwind flux is

replaced by the downwind flux in the adjoint equation.

In the decomposition of A (4.51), the coefficient matrices C_1 and C_2 are left-multiplied, but in the decomposition of A^T (4.59), C_1 and C_2 are right-multiplied. The difference is: left-multiplying a diagonal matrix scales the rows of the operated matrix, while the right-multiplication scales the columns of the operated matrix.

We first evaluate the multiplication of coefficient matrices on the volume terms. From forward discretization to the adjoint discretization, the volume coefficients of the pressure and velocity fields are swapped. Taking strong form formulation for example, we have $\rho_k c_k^2 (\nabla \cdot \mathbf{v}, \varphi)_{D^k}$ and $\rho_k^{-1} \left(\frac{\partial p}{\partial x_i}, \varphi \right)_{D^k}$ in the forward problem, but use $\rho_k^{-1} (\nabla \cdot \mathbf{v}, \varphi)_{D^k}$ and $\rho_k c_k^2 \left(\frac{\partial p}{\partial x_i}, \varphi \right)_{D^k}$ in the adjoint problem.

The impact of multiplying coefficient matrices on the surface terms is more complicated. First, like the volume terms, the coefficients ρc^2 and ρ^{-1} are swapped in the central flux, but remain the same in the penalty term. Second, the model parameters may also be taken as numerical fluxes in the surface contribution, and the adjoint of them must take the difference between left-multiplying a matrix and right-multiplying a matrix into consideration. Taking a model parameter m and a solution field u as an example, we may have different $m^* \llbracket u \rrbracket$ or $m^* \{\{u\}\}$ in the forward problem, such as

$$m^- \llbracket u \rrbracket, \quad m^- \{\{u\}\}, \quad (4.60a)$$

$$\frac{m^- + m^+}{2} \llbracket u \rrbracket, \quad \frac{m^- + m^+}{2} \{\{u\}\}, \quad (4.60b)$$

$$\frac{2m^- m^+}{m^- + m^+} \llbracket u \rrbracket, \quad \frac{2m^- m^+}{m^- + m^+} \{\{u\}\}. \quad (4.60c)$$

In the adjoint equation, they become

$$m^+u^+ - m^-u^-, \quad \frac{m^+u^+ + m^-u^-}{2}, \quad (4.61a)$$

$$\frac{m^- + m^+}{2} \llbracket u \rrbracket, \quad \frac{m^- + m^+}{2} \{\{u\}\}, \quad (4.61b)$$

$$\frac{2m^-m^+}{m^- + m^+} \llbracket u \rrbracket, \quad \frac{2m^-m^+}{m^- + m^+} \{\{u\}\}, \quad (4.61c)$$

respectively.

The difference between the forward and adjoint formulations also explains why we are solving a different PDE (4.31) in the adjoint simulation at the continuous level.

Recall that we now have the fully-discrete formulation for both forward problem (4.34) and adjoint problem (4.49). If we define operator

$$L[\mathbf{u}^n] = M \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \sum_{i=0}^2 \alpha_i A \mathbf{u}^{n-i}, \quad (4.62a)$$

$$L^*[\boldsymbol{\lambda}^n] = M \frac{\boldsymbol{\lambda}^{n-1} - \boldsymbol{\lambda}^n}{\Delta t} + \sum_{i=0}^2 \alpha_i A^T \boldsymbol{\lambda}^{n+i}. \quad (4.62b)$$

The following relation must hold,

$$\sum_{n=0}^N \langle L[\mathbf{u}^n], \boldsymbol{\lambda}^n \rangle = \sum_{n=0}^N \langle L^*[\boldsymbol{\lambda}^n], \mathbf{u}^n \rangle, \quad (4.63)$$

where the bracket represents vector-vector inner product, \mathbf{u}^n are any class of vectors that satisfy $\mathbf{u}^{-2} = \mathbf{u}^{-1} = \mathbf{u}^0 = 0$, and $\boldsymbol{\lambda}^n$ are any class of vectors with terminal condition $\boldsymbol{\lambda}^N = \boldsymbol{\lambda}^{N+1} = \boldsymbol{\lambda}^{N+2} = 0$.

An adjoint test can be conducted to verify the adjoint solver. In the adjoint test, we generate a set of random vectors of \mathbf{u}^n and $\boldsymbol{\lambda}^n$ with the initial and terminal constraints on \mathbf{u}^n and $\boldsymbol{\lambda}^n$. We then evaluate the relative difference between the inner

products by using the following expression,

$$\left| \frac{\sum_{n=0}^N \langle L[\mathbf{u}^n], \boldsymbol{\lambda}^n \rangle - \sum_{n=0}^N \langle L^*[\boldsymbol{\lambda}^n], \mathbf{u}^n \rangle}{\sum_{n=0}^N \langle L[\mathbf{u}^n], \boldsymbol{\lambda}^n \rangle} \right|. \quad (4.64)$$

If our derivation and implementation are correct, this value should be around machine precision.

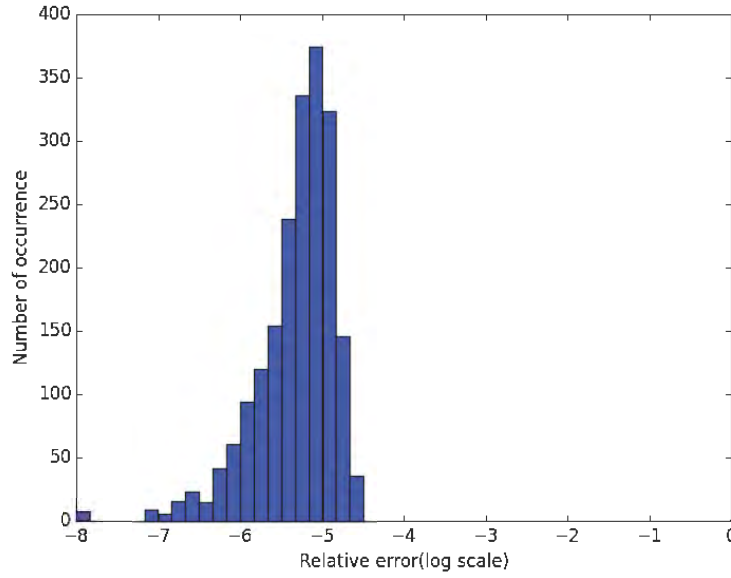


Figure 4.1: The distribution of the relative difference in adjoint test. The x -axis is the relative difference in log scale and the y -axis is the number of occurrence in the experiment.

In our experiment, we conducted an adjoint test on GL-HEX elements. In the formulation, we take $c^* = c^-$ for model parameters in the forward run, and use first order absorbing boundary condition. We set the total number of time steps $N = 100$, the total number of elements $K = 64$, \mathbf{u}^n and $\boldsymbol{\lambda}^n$ are randomly generated vectors, and the velocity model is also randomly generated. The experiment was repeated 2,000 times using single precision, and the relative differences (4.64) were recorded. Figure 4.1 shows the distribution of the relative difference, where the x -axis indicates the relative error in log scale, and y -axis is the number of occurrence in the experiment. As

we can see in the histogram, we obtain 5 to 6 digits of accuracy most of the time. The reason we do not always get exact machine precision is due to the condition numbers of the forward and adjoint operators which harm the accuracy of the evaluation.

4.3.3 The gradient

According to our derivation, the actual gradient of the misfit function is given in (4.50) as

$$\frac{d\mathcal{J}}{d\mathbf{m}} = \Delta t \sum_{n=0}^N (\boldsymbol{\lambda}^n)^T \frac{d\mathbf{h}^n}{d\mathbf{m}}, \quad (4.65)$$

where $\boldsymbol{\lambda}^n$ is the solution to the adjoint equation, and \mathbf{h}^n is the forward problem defined in (4.40). As we have already discussed the adjoint equation in the previous subsection, the only thing left to write down the explicit form of $d\mathbf{h}^n/d\mathbf{m}$.

Recall that \mathbf{h}^n is defined by

$$\mathbf{h}^n := M \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \sum_{i=0}^2 \alpha_i A \mathbf{u}^{n-i} - \mathbf{f}^n, \quad (4.66)$$

and the model parameters only exist in matrix A . Therefore,

$$\frac{d\mathbf{h}^n}{d\mathbf{m}} = \sum_{i=0}^2 \alpha_i \frac{d(A\mathbf{u}^{n-i})}{d\mathbf{m}}. \quad (4.67)$$

Using the same decomposition of A as we did before, we have $A = C_1 D + C_2 S$, where C_1 and C_2 are coefficient matrix containing model parameters as defined in (4.52) and (4.53).

For simplicity, we take the flux on the coefficient as $c^* = c^-$ and $\rho^* = \rho^-$. The

explicit form of $d(\mathbf{A}\mathbf{u})/d\mathbf{m}$ can be easily write down as

$$\frac{\partial(\mathbf{A}\mathbf{u})}{\partial c} = \frac{\partial C_1}{\partial c} D\mathbf{u} + \frac{\partial C_2}{\partial c} S\mathbf{u}, \quad (4.68)$$

$$\frac{\partial(\mathbf{A}\mathbf{u})}{\partial \rho} = \frac{\partial C_1}{\partial \rho} D\mathbf{u} + \frac{\partial C_2}{\partial \rho} S\mathbf{u}, \quad (4.69)$$

where $\partial C_i/\partial c$ and $\partial C_i/\partial \rho$ are entry-wise partial derivative and are given by

$$\frac{\partial C_1}{\partial c} = \begin{bmatrix} 2c_1 & & & & \\ & \ddots & & & \\ & & 2c_K & & \\ & & & 0 & \\ & & & & \ddots \\ & & & & & 0 \end{bmatrix}, \quad (4.70)$$

$$\frac{\partial C_2}{\partial c} = I, \quad (4.71)$$

$$\frac{\partial C_1}{\partial \rho} = \begin{bmatrix} c_1^2 & & & & \\ & \ddots & & & \\ & & c_K^2 & & \\ & & & -\rho_1^{-2} & \\ & & & & \ddots \\ & & & & & -\rho_K^{-2} \end{bmatrix}, \quad (4.72)$$

$$\frac{\partial C_2}{\partial \rho} = 0. \quad (4.73)$$

With the above formulas, we have derived the explicit formula for the gradient of the misfit function using discrete adjoint-state method.

Here we assume $c^* = c^-$ and $\rho^* = \rho^-$ in the forward problem. If one takes more complicated fluxes for the model parameters in the forward problem, the explicit

formula for the gradient can be more complicated.

Equation (4.65) gives the formula to obtain the FWI gradient using DG discretization. Theoretically, this approach can result in a discretely exact gradient up to machine precision. However, in our implementation, we do not follow this theoretical derivation, and our gradient contains numerical errors. The difficulties for computing discretely exact gradient in our implementation are:

- Regenerating forward wavefield in backward stage introduces errors. As discussed in section 3.3, we do not save the forward wavefield in memory at forward phase stage. Instead, we save only the boundary values of the forward wavefield, and provide it as the boundary condition to regenerate the forward wavefield in the backward run. Although the two wavefields are mathematically identical, they are not numerically the same. Extra numerical errors are introduced in the regenerated forward wavefield, and hence in equation (4.65), we do not have exact $d\mathbf{h}^n/d\mathbf{m}$.
- Seismic data is interpolated to fit the time grid. The adjoint equation (4.49) suggests that the data residue should be injected through L^2 projection. However, in our DG solver, we use a scattered-total field formulation for source injection (see section 3.1.2). Although the scattered-total field formulation avoids the numerical error introduced by the L^2 projection, it requires our knowledge of the source signal at any time points. However, this is not available in the discretely recorded data. In the implementation, we use a polynomial interpolation to obtain the continuous data in the timeline, and this introduces an interpolation error.

Although our gradient is not exact up to machine precision, it is still a good approximation to the “true” gradient. As long as it is a descent direction, the ap-

proximated gradient can be used in the steepest descent method to obtain convergence [138].

4.4 Numerical results of conventional FWI

In this section, we provide the numerical results of the FWI scheme, and recover the true velocity model with a decent initial guess. We test our FWI scheme on three different models: (1) a Gaussian inclusion model; (2) a cubic inclusion model; (3) a layer model.

4.4.1 Gaussian inclusion model

We test the FWI implementation on a model where a Gaussian inclusion is centered in the domain. Due to the smoothness of the model, a large portion of the energy is conveyed by the transmitted waves, so we first place the receivers at the bottom of the domain.

The configuration is: the computational domain is a cube of size $[0\text{km}, 2\text{km}]^3$; in the true model, a Gaussian inclusion is centered in the domain with a velocity ranging from 1.0 km/s to 1.1 km/s (figure 4.2); the initial model is a constant velocity model with a background velocity of 1.0 km/s (figure 4.3); 5 source signals are injected on plane $z = 0.15$ km, which are Ricker pulses of 3 Hz peak frequency; 729 receivers are evenly distributed on plane $z = 1.75$ km (figure 4.4); terminating time is $T = 3.0$ s; the mesh used for the test is a hexahedral mesh with a grid size of $20 \times 20 \times 20$.

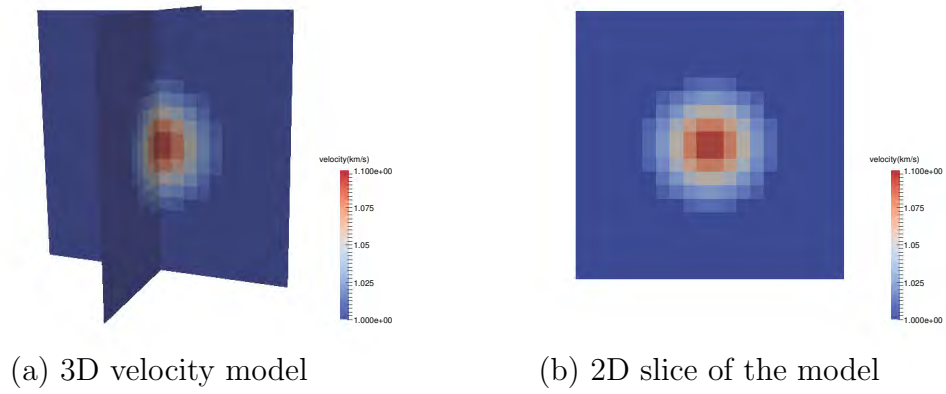


Figure 4.2: Gaussian inclusion example: true velocity model.

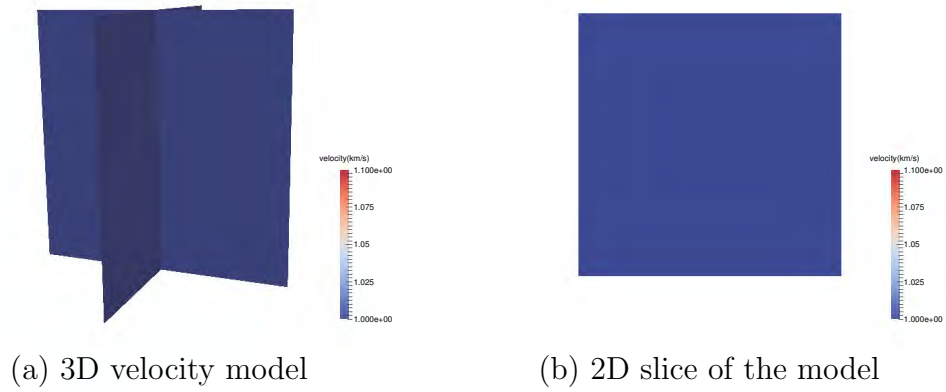


Figure 4.3: Gaussian inclusion example: initial velocity model.

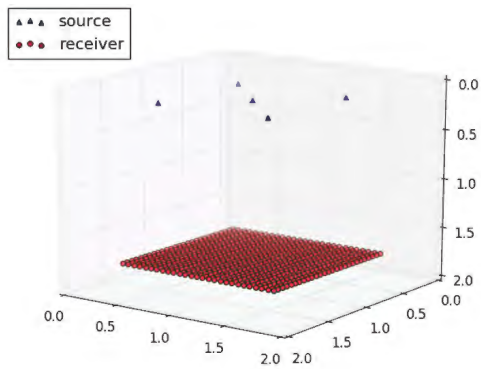


Figure 4.4: Gaussian inclusion example: sources and receivers, where receivers are buried in the bottom of the domain.

The results of this experiment are plotted in figure 4.5 and figure 4.6. Figure 4.5 reports the resulted model from the FWI iterations in the same color scale with the true model. The solution is getting close to the true model from the initial model. Figure 4.6 is the history of the misfit values. The maximum number of iterations is set to 50, and the final misfit ends up to be 0.04% of the original misfit.

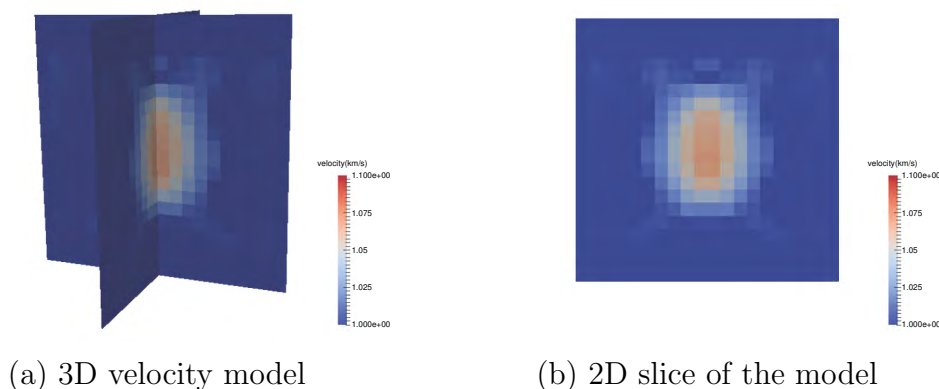


Figure 4.5: Gaussian inclusion example: FWI results with receivers buried in the bottom of the domain.

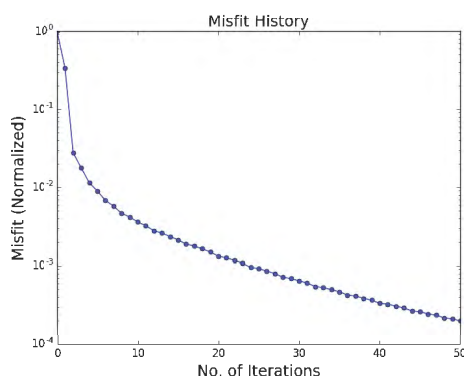


Figure 4.6: Gaussian inclusion example: history of misfit values with receivers buried in the bottom of the domain. The misfit value drops to 0.04% after 50 iterations.

Due to ill-posedness of FWI and insufficient data coverage, the FWI solution still has some artifacts (figure 4.5). One way to improve this is to add more receivers at

different locations to collect more information of the velocity model. As illustrated in figure 4.7, 4374 receivers are located near all boundary faces in the domain. The final misfit after 50 iterations is 0.02%. The result generated from this configuration is plotted in figure 4.8 and figure 4.9. The resulted velocity model is closer to the true velocity model than the previous one.

The additional receivers improve the data coverage. As a result, the model is better inverted. In real applications, some additional data or information may be obtained from sonic logs or exploration well. For example, Asnaashari et al. [139] used sonic logs as a prior model for FWI to improve the results.

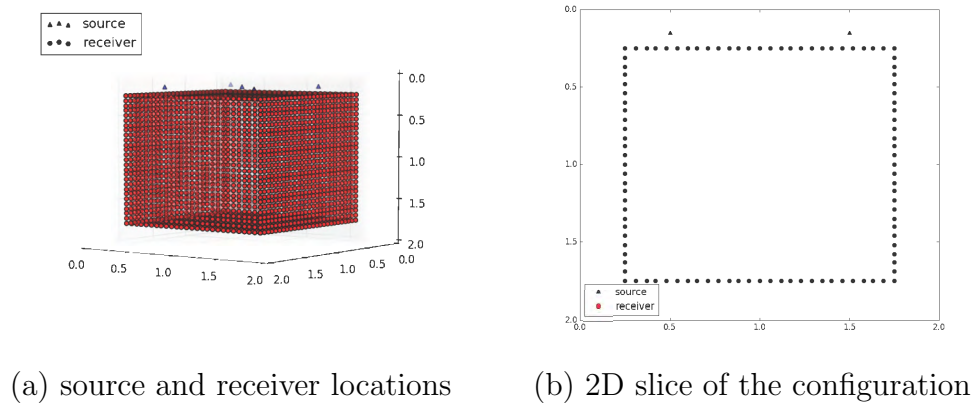


Figure 4.7: Gaussian inclusion example: sources and receivers, where receivers are located near all boundary faces.

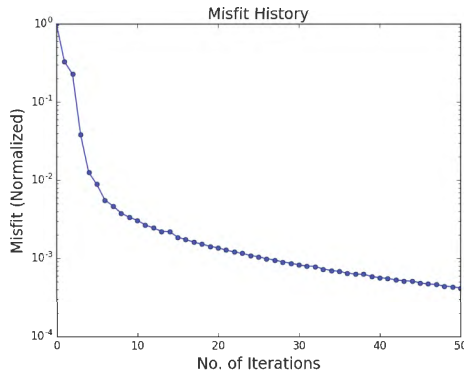


Figure 4.9: Gaussian inclusion example: history of misfit values with receivers located near all boundary faces. The misfit value drops to 0.02% after 50 iterations.

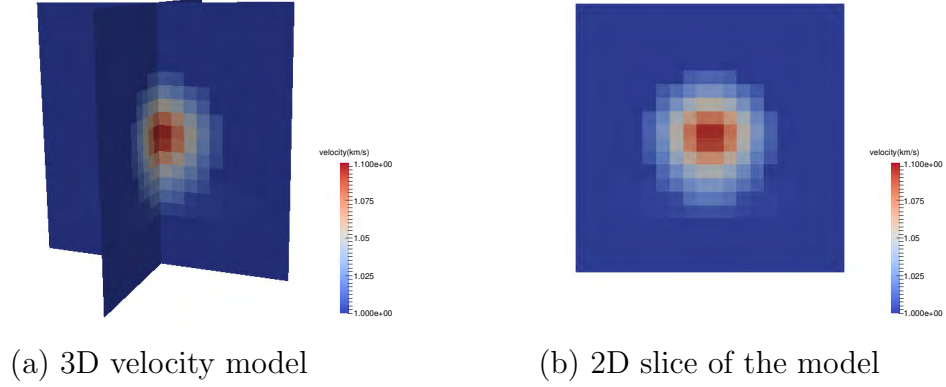


Figure 4.8: Gaussian inclusion example: FWI results with receivers located near all boundary faces.

4.4.2 Cubic inclusion model

We test a model with a centered cubic inclusion. Due to the sharp velocity interface, reflected waves can be observed in the simulation. At the same time, a significant portion of energy resides in the transmitted waves. Therefore, we put receivers on both top and bottom of the domain to collect enough data.

The velocity inside the inclusion is a constant. The configuration is: the computational domain is a cube of size $[0\text{km}, 2\text{km}] \times [0\text{km}, 2\text{km}] \times [0\text{km}, 2\text{km}]$; in the true model, a cubic inclusion, which has side length of 0.8 km and velocity of 1.2 km/s, is centered in the domain; the velocity outside the inclusion is 1.0 km/s (figure 4.10); the initial model is a constant velocity model with a background velocity of 1.0 km/s (figure 4.11); 5 source signals are injected on plane $z = 0.15$ km, which are Ricker pulses of 3 Hz peak frequency; 729 receivers are evenly distributed on plane $z = 0.25$ km, and another 729 receivers are buried at $z = 1.75$ km (figure 4.12); terminating time is $T = 3.0$ s; the mesh used for the test is a hexahedral mesh with a grid size of $20 \times 20 \times 20$.

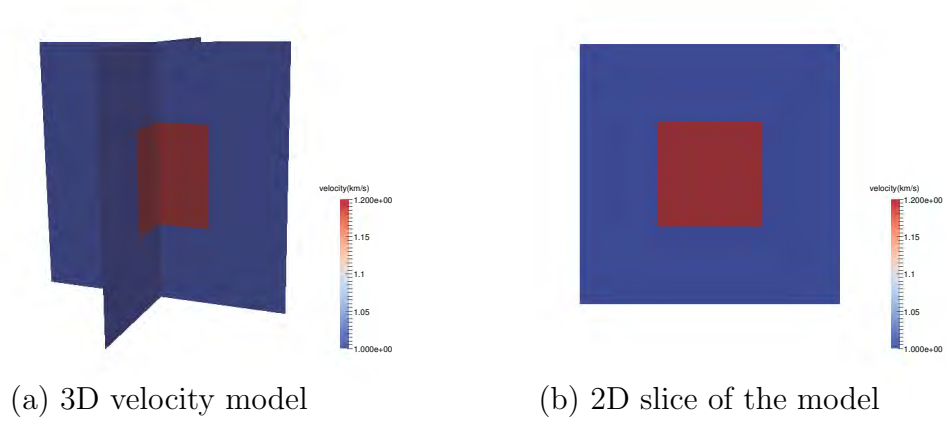


Figure 4.10: Cubic inclusion example: true velocity model.

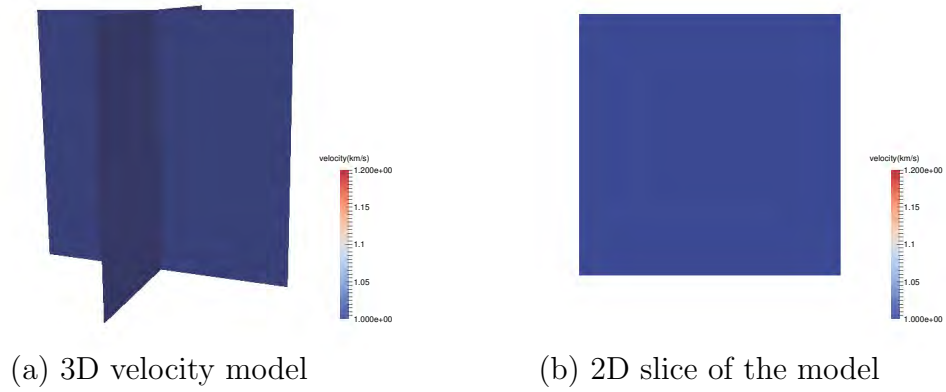


Figure 4.11: Cubic inclusion example: initial velocity model.

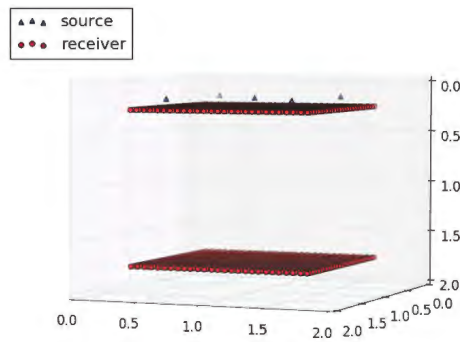


Figure 4.12: Cubic inclusion example: sources and receivers, where receivers are placed near top and bottom boundary faces.

Figure 4.13 and figure 4.14 are the FWI results under this configuration. In figure 4.13, the velocity model is mostly recovered with some artifacts at the bottom of the velocity interfaces. Figure 4.14 is the history of the misfit values throughout the FWI iterations, the misfit decreases to 0.09% after 100 iterations.

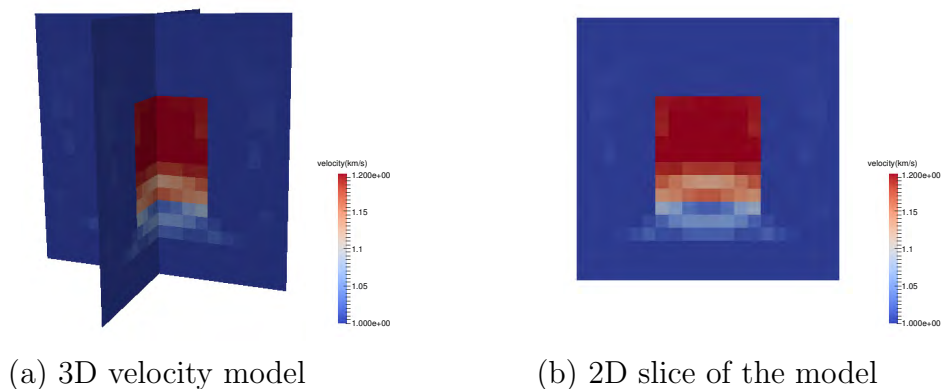


Figure 4.13: Cubic inclusion example: FWI results.

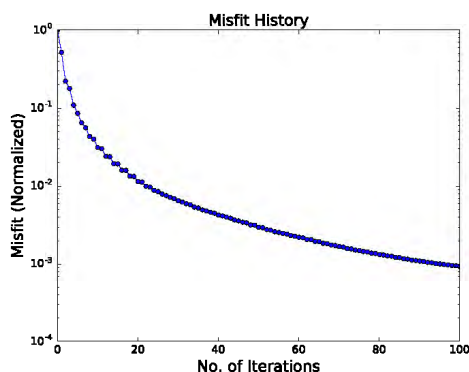


Figure 4.14: Cubic inclusion example: history of misfit values. The misfit value drops to 0.09% after 100 iterations.

4.4.3 Layer model

In this subsection, we test a layer model. A sharp interface and large velocity contrast lead to significant reflected waves, and we use the receivers on the top of the domain to collect the data.

The configuration is: the computational domain is a cube of size $[0\text{km}, 2\text{km}] \times [0\text{km}, 2\text{km}] \times [0\text{km}, 2\text{km}]$; in the true model, the velocity in bottom part of the model ($\{1\text{km} < z \leq 2\text{km}\}$) is 1 km/s, and the velocity in top part of the model ($\{0\text{km} \leq z \leq 1\text{km}\}$) is 2 km/s (figure 4.15); the initial model is a smooth version of the true model (figure 4.16); 9 source signals are injected on plane $z = 0.15$ km, which are Ricker pulses of 2 Hz peak frequency; 729 receivers are evenly distributed on plane $z = 0.25$ km (figure 4.17); terminating time is $T = 3.0$ s; the mesh used for the test is a hexahedral mesh with a grid size of $20 \times 20 \times 40$.

To better invert the model, some techniques are applied to the inversion. First, we set a water layer above $z = 3.0$ km (first 6 cells of the grid in z -direction), and the velocity in the water layer is invariant. Second, we set a box constraint on the velocity model, which is $1\text{km/s} \leq c \leq 2\text{km/s}$, and use projected gradient method to project the velocity model back to the desired range.

The box constraint does not only restrict the feasible region, but also guarantee that the CFL condition will not be violated. Otherwise, the numerical solution to the wave equations may blow up after a certain number of FWI iterations.

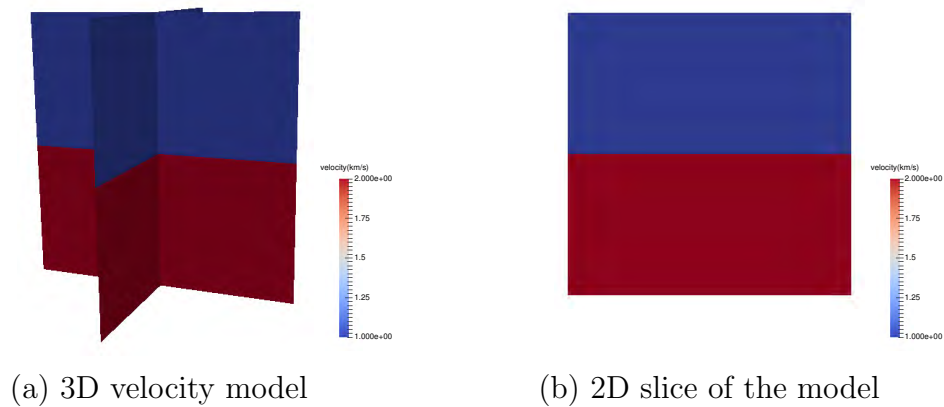


Figure 4.15: Layer model example: true velocity model.

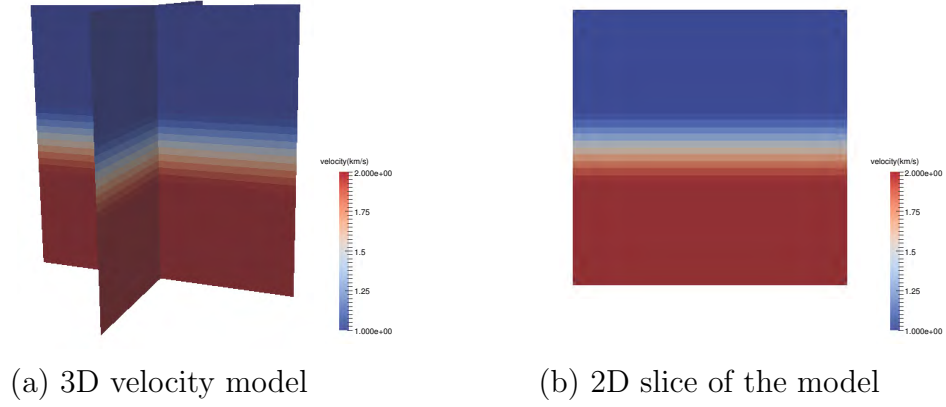


Figure 4.16: Layer model example: initial velocity model.

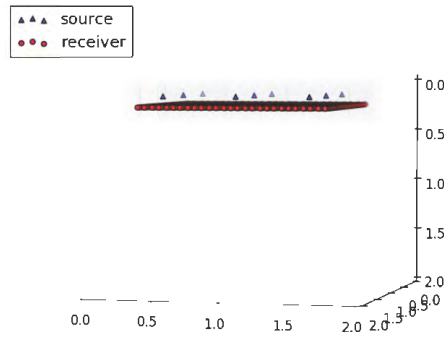


Figure 4.17: Layer model example: sources and receivers, where receivers are placed near the top to receive reflected waves.

Figure 4.18 and figure 4.19 are the FWI results of the layer model. Due to the difficulty of the inverse problem, the model is not fully recovered. In figure 4.18, the velocity model near the interface becomes sharper, but we can not tell where the media interface exactly is. Figure 4.19 is the history of the misfit values throughout the FWI iterations. Different from the previous test cases where the misfit values drop down quickly, the misfit value in this experiment only decreases to 2.28% after 150 iterations.

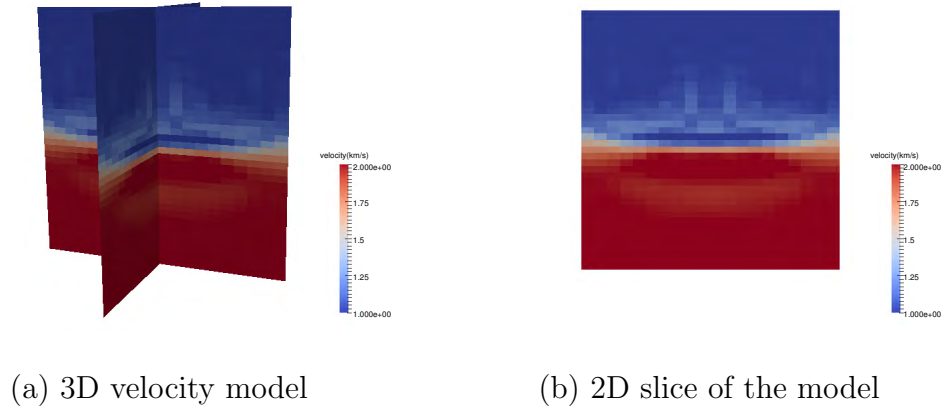


Figure 4.18: Layer model example: FWI results.

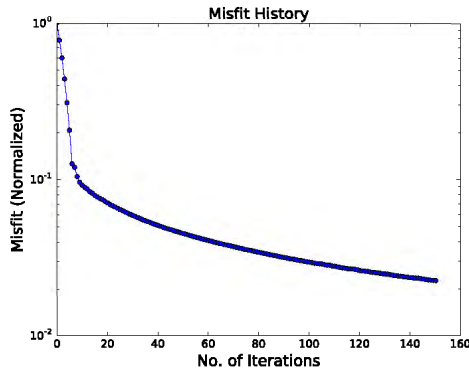


Figure 4.19: Layer model example: history of misfit values. The misfit value drops to 2.28% after 150 iterations.

4.5 Inverting sharp interfaces

Compared to finite difference method, DG has a higher order interface error [26]. Therefore, recovering the location of media interfaces is crucial for DG-FWI.

Conventional FWI is capable of inverting smooth velocity models, but not good at inverting sharp interfaces. In recent researches, additional features were added to conventional FWI to invert sharp models.

One approach is adding a penalty term to the conventional FWI formulation.

Typically, an L^1 like penalty of the model gradient (sometimes known as total variation penalty) can create sparsity of the model gradient and hence introduce sharpness in the result [140]. An example of this approach can also be found in Esser et al. [141]. Although sharpness can be introduced to the FWI results, this approach does not explicitly tell the location of media interfaces. One might still have to manually pick the interfaces, which damages the accuracy of the solution.

Another approach, known as the level set method, can be used for solving inverse problems involving obstacles [142], and is recently applied to invert salt bodies in FWI [143, 144, 145]. The level set method evolves a level set function, which can nicely capture the evolution of the media interfaces including the topology changes. The level set function is solved by a Hamilton-Jacobi equation. However, the Hamilton-Jacobi equation is challenging to solve, which makes the level set method complicated. The level set method is typically implemented with specifically designed numerical schemes, re-initialization strategy, and penalization terms [144].

In this thesis, we propose a new approach to recovering the media interfaces. Our method is inspired by the level set method, but much simpler than the level set method. It explicitly gives the location of the interfaces, and hence we are able to generate aligned meshes. Our approach resembles the shape optimization proposed by Schmidt et al. [146], but we use a different formulation and have specific deformations of the shape.

4.5.1 Perturb surfaces along the normal directions

Assume we have an initial guess of inclusion (or interfaces) of the velocity model, and we want to perturb the geometry of this inclusion towards the true velocity model.

Recall that in the derivation of the continuous adjoint state method, we have the

impact of model perturbation (4.15) represented as

$$\mathcal{J}[m + \delta m] - \mathcal{J}[m] = \langle \delta m, F^* S^*(S\mathcal{F}[m] - \mathbf{d}) \rangle + \mathcal{O}(\|\delta m\|^2), \quad (4.74)$$

where \mathcal{F} is the forward operator, S is the point-wise restriction operator, \mathbf{d} is the data residue, F^* is the adjoint of the Born operator F , and S^* is the adjoint of S . Denoting image I as

$$I(\mathbf{x}) = F^* S^*(S\mathcal{F}[m] - \mathbf{d}), \quad (4.75)$$

equation (4.74) can be simplified to

$$\mathcal{J}[m + \delta m] - \mathcal{J}[m] = \int_{\Omega} \delta m(\mathbf{x}) I(\mathbf{x}) dx + \mathcal{O}(\|\delta m\|^2). \quad (4.76)$$

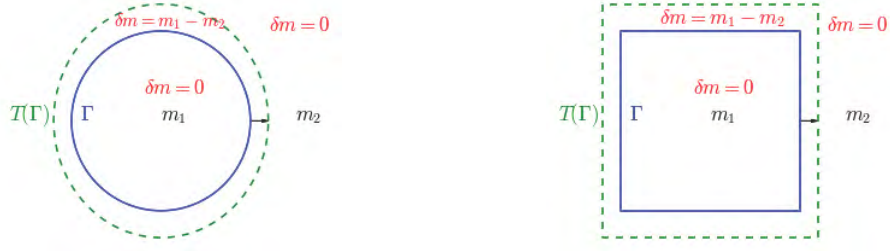
Now, we perturb the surface of the inclusion boundary along the normal directions, and each point \mathbf{x} on the inclusion boundary Γ is moved for a distance h . Figure 4.20 illustrates this idea: the velocity model is m_1 inside the inclusion and m_2 outside the inclusion; the blue line Γ is the original boundary of the inclusion; the green line $T(\Gamma)$ is the perturbed boundary of the inclusion; velocity change is 0 inside Γ and outside $T(\Gamma)$ and $m_1 - m_2$ between Γ and $T(\Gamma)$. Assuming the perturbation is small, we can rewrite (4.76) as

$$\mathcal{J}[m + \delta m] - \mathcal{J}[m] \approx (m_1 - m_2) \int_{\Gamma} h I(\mathbf{x}) d\tau. \quad (4.77)$$

To obtain a derivative, we let h take the limit to zero, and hence we have

$$\frac{\partial \mathcal{J}}{\partial r} = (m_1 - m_2) \int_{\Gamma} I(\mathbf{x}) d\tau, \quad (4.78)$$

where r is the position coordinate of a surface along its normal direction. Expression



(a) perturbation of a circle

(b) perturbation of a polygon

Figure 4.20: Perturbation of the inclusion boundary along the normal directions. The impact of this perturbation is represented by the area between Γ and $T(\Gamma)$.

(4.78) is the derivative of the misfit function with respect to the perturbation of a surface along its normal direction. Since conventional FWI already computes the imaging condition $I(\mathbf{x})$, the workflow of this derivative computation in DG can be summarized in algorithm 12.

Algorithm 12: Compute derivative w.r.t. cube inclusion radius

Input: a velocity model m and an inclusion boundary Γ

Output: derivative of misfit function w.r.t. cube inclusion radius $\partial\mathcal{J}/\partial r$

- 1 Apply the adjoint-state method (i.e. RTM) to get the image $I(\mathbf{x})$ of the current model m .
 - 2 Identify the set of element surfaces which are also the boundary surfaces of the cube inclusion.
 - 3 Set $\partial\mathcal{J}/\partial r = 0$.
 - 4 **for** each face $\partial D^k \subset \Gamma$ **do**
 - 5 Add the contribution of the surface integration to $\partial\mathcal{J}/\partial h$ by

$$\frac{\partial\mathcal{J}}{\partial r} += (m_1 - m_2) \int_{\partial D^k} \frac{I^+(\mathbf{x}) + I^-(\mathbf{x})}{2} d\tau.$$
 - 6 Return $\partial\mathcal{J}/\partial r$.
-

4.5.1.1 Cubic inclusion example

We study an experiment with a cubic inclusion in the center of the domain. The radius of the cubic inclusion (i.e. half side length) is to be solved. We can perturb the radius of the cube to invert the velocity model.

The configuration is: the computational domain is a cube of size $[0 \text{ km}, 2 \text{ km}]^3$; a cubic inclusion is centered at $(1 \text{ km}, 1 \text{ km}, 1.25 \text{ km})$; in the true velocity model, the velocity is 1.2 km/s inside the inclusion and 1.0 km/s outside the inclusion; the true radius of the inclusion is $r = 0.3 \text{ km}$; one shot is injected at $(1 \text{ km}, 1 \text{ km}, 0.35 \text{ km})$ as source, which is a Ricker pulse of 3 Hz peak frequency; 729 receivers are evenly distributed on plane $z = 0.5 \text{ km}$; recording end time is $T = 2.4 \text{ s}$. Figure 4.21 illustrates the configuration of this experiment.

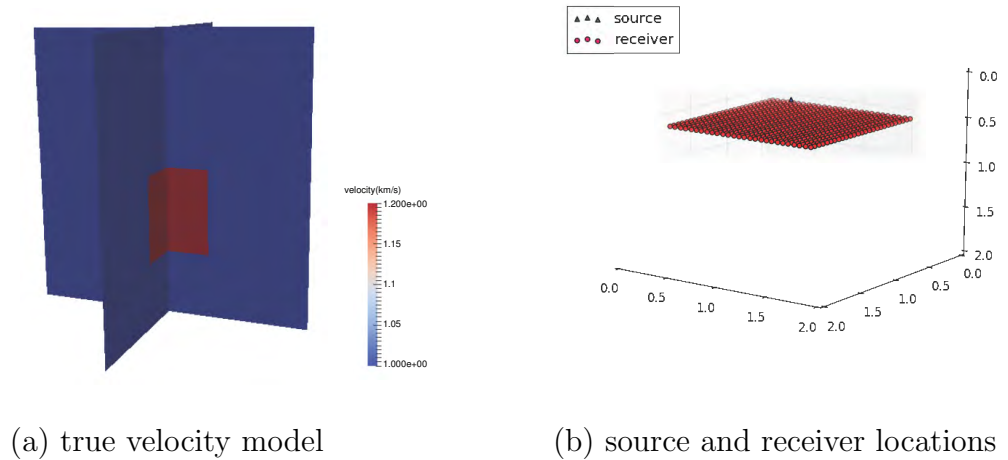


Figure 4.21: Configuration for cubic inclusion example: (a) true velocity model, where a cubic inclusion is located near the center of the domain; (b) the locations of sources and receivers: the blue triangle is the location of the source and the red dots are the location of the receivers.

First, to start with a simple case, we fix the the velocity inside and outside the inclusion to be same as the true values, but the radius of the inclusion can be perturbed. Therefore, the inversion degenerates to a 1D problem, and only the inclusion radius

needs to be inverted. We let the the radius inclusion vary among a series of values near its true value, and plot the misfit function and its corresponding derivatives in figure 4.22. Figure 4.22 (a) is the misfit value with the variation of the inclusion radius. Figure 4.22 (b) compares the derivatives computed by the surface integration of the imaging condition and the finite difference approach. As a result, the derivative computed by surface integration matches the reference derivative (computed by finite difference), and hence it validates the correctness of our algorithm.

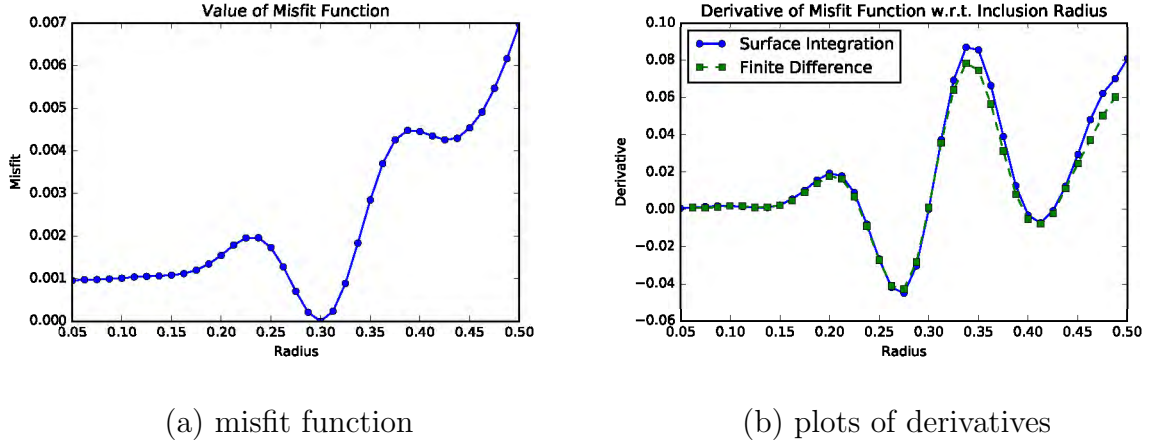


Figure 4.22: Misfit function and its derivative w.r.t to inclusion radius r : (a) value of misfit function; (b) derivative of misfit function w.r.t. inclusion radius r , where the blue line is the derivative computed using surface integration in algorithm 12 and the green line is computed by finite difference.

When the media interfaces move (i.e. the inclusion radius changes), we need to regenerate mesh to make sure that the mesh is aligned with the interfaces. To this end, we use a `Gmsh` script to regenerate mesh whenever the interfaces are perturbed. Figure 4.23 is a mesh example used in our experiments, which contains only hexahedral elements.

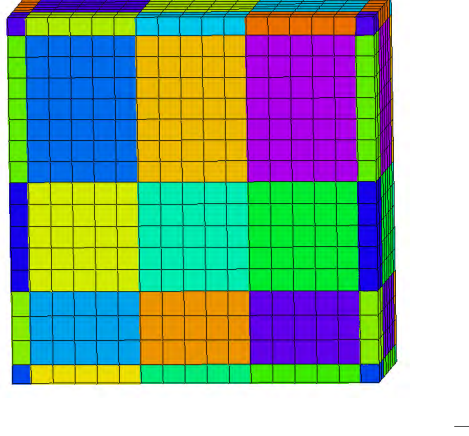
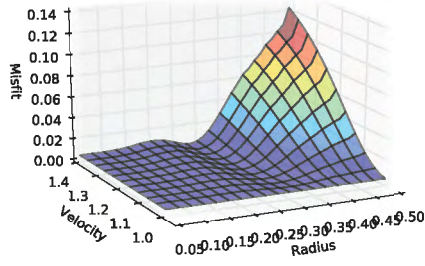


Figure 4.23: Mesh example for cubic inclusion, where the mesh is aligned with the inclusion boundary.

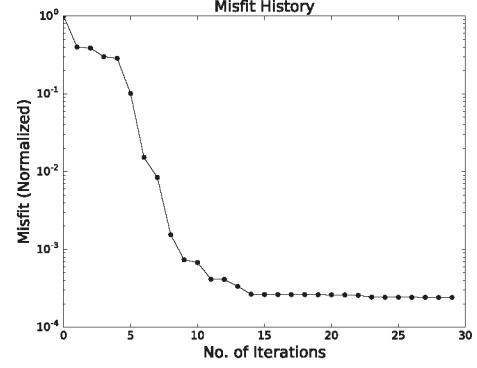
Next, we study an inversion of two unknown parameters on this model. Here, the inclusion radius and the velocity inside the inclusion are the parameters to be inverted. Recall that the true values are: radius $r_{\text{true}} = 0.3$ km, and inclusion velocity $c_{\text{true}} = 1.2$ km/s. We set the initial guess to: radius $r_{\text{init}} = 0.25$ km, and inclusion velocity $c_{\text{init}} = 1.1$ km/s.

When we update the inclusion radius, a new mesh need to be regenerated. When we update the velocity inside the inclusion, there is no need to regenerate mesh. Taking this into consideration, we apply an alternative updating scheme: updating the radius and velocity alternatively in the FWI iterations.

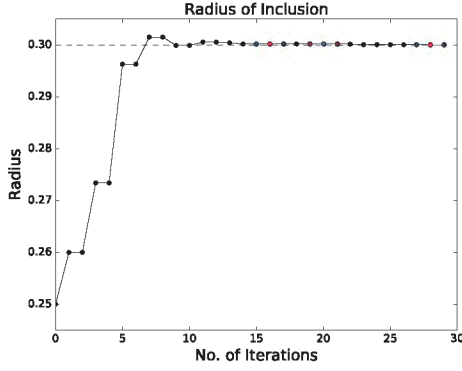
Figure 4.24 presents the results of the inversion. Figure 4.24 (a) plots the value of the misfit function with different inclusion radius and velocity. Figure 4.24 (b) (c) and (d) show the history of the misfit function, inclusion radius and inclusion velocity respectively, where the red markers correspond to the radius update and the blue markers correspond to the velocity update. The dash line in figure 4.24 (c) and (d) are the true values of the model.



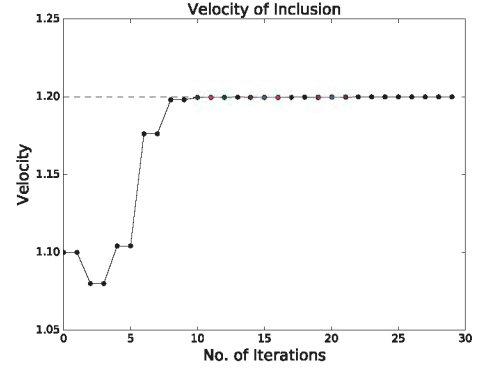
(a) misfit function



(b) misfit history



(c) history of inclusion radius



(d) history of inclusion velocity

Figure 4.24: Experiment of alternating update: (a) value of misfit function; (b) history of misfit function, where the red and blue markers correspond to update of radius and velocity respectively; (c) history of inclusion radius r , where $r_{\text{true}} = 0.3$ km; (d) history of inclusion velocity c , where $c_{\text{true}} = 1.2$ km/s.

In the result, our model converges to the true model and the misfit function converges to zero. This indicates our algorithm does work for the simple cubic inclusion example.

4.5.1.2 Multi-surface example

In the previous example, there is only one parameter (the inclusion radius) to be inverted using the surface integration. To better verify our algorithm, we design a multi-surface example, where several media interfaces are mis-located. The gradient

used for surface perturbation is then obtained by computing the surface integration on individual surfaces.

To simplify the computation, we now use a 2.5D model, where the model in y -axis is short and set to a constant. Therefore, we are only interested in the projection of the model on x - z plane.

The configuration is: the computational domain is a cube of size $[0 \text{ km}, 2 \text{ km}] \times [0 \text{ km}, 0.3 \text{ km}] \times [0 \text{ km}, 2 \text{ km}]$; in the true velocity model, the velocity is 5.0 km/s inside the bottom inclusion and 1.0 km/s outside the inclusion; three source signals are injected on plane $z = 0.2 \text{ km}$, which are Ricker pulses of 3 Hz peak frequency; 27 receivers are evenly distributed on plane $z = 0.35 \text{ km}$; recording end time is $T = 3.0 \text{ s}$. Figure 4.25 also presents the configuration of this experiment.

In the FWI iterations, we keep regenerating meshes to align the mesh with the media interfaces. An example of the mesh is shown in figure 4.26.

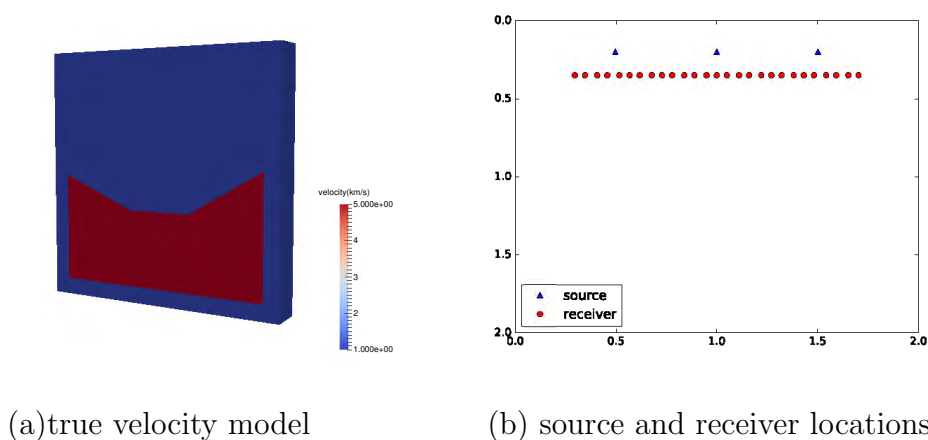


Figure 4.25: Configuration for multi-surface example: (a) true velocity model, where a ployhedron is located at the bottom of the domain with its top surfaces to be inverted; (b) the locations of sources and receivers: the blue triangle is the location of the source and the red dots are the location of the receivers.

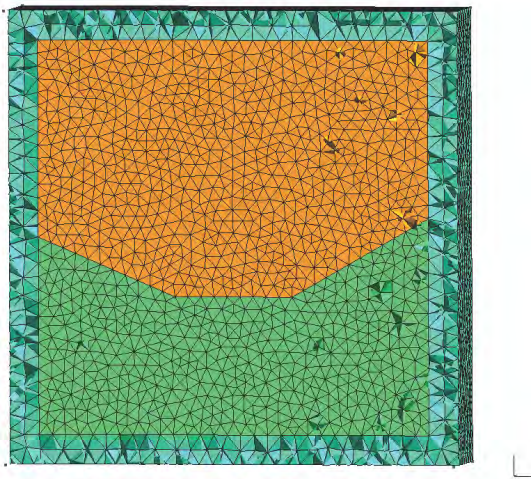


Figure 4.26: Mesh for multi-surface example, where the mesh is aligned with the media interfaces.

We then solve the inverse problem using the surface integration technique. Figure 4.27 plots the initial and true models of the inclusion, where the top surfaces are mislocated. In the experiment, there are three degrees of freedom in the model space: each top surface can be perturbed along its normal direction. Figure 4.28 shows the result of our FWI experiment. In figure 4.28 (a) we can see that the solution overlaps with the true model. In addition in the misfit history plotted in figure 4.28 (b), the misfit value converges to zero.

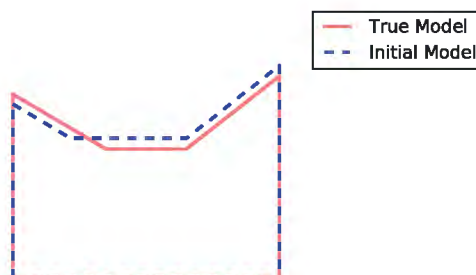


Figure 4.27: True and initial models of the multi-surface example.

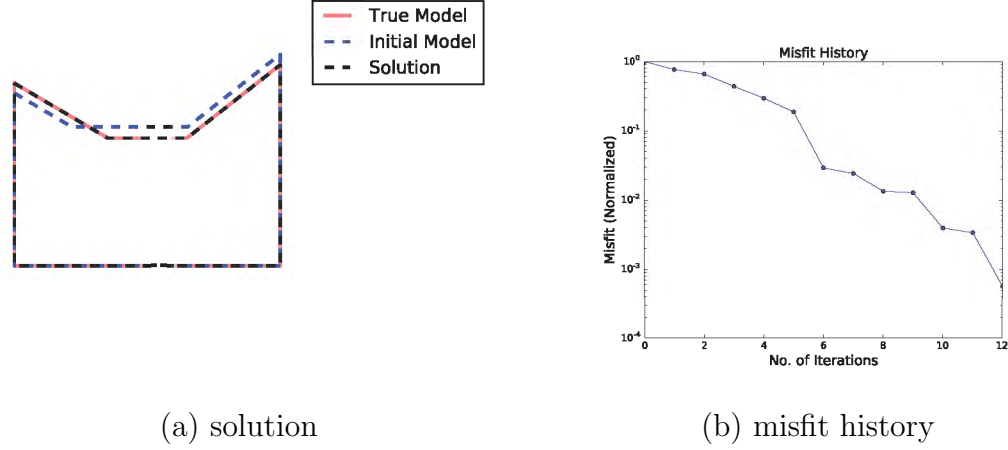


Figure 4.28: Solution of the multi-surface example: (a) geometry of the solution; (b) misfit history of the FWI iterations. The misfit value drops below 1‰ after 12 iterations.

This multi-surface example shows that we are able to perturb many surfaces simultaneously along their normal directions. However, this type of perturbation requires our knowledge of the normal directions of the media interfaces. The update of the velocity model does not change the normal directions of the interfaces. This restriction can be relaxed by introducing more types of perturbations as we will discuss in the next subsection.

4.5.2 Perturb vertices along a given direction

Assume the inclusions/layers in our velocity model are made up of planar surfaces. Using the same idea as that in the previous subsection, we can perturb the vertices in the FWI update. Recall that the impact of the model perturbation is given by equation (4.76). We rewrite it here as

$$\mathcal{J}[m + \delta m] - \mathcal{J}[m] = \int_{\Omega} \delta m(\mathbf{x}) I(\mathbf{x}) d\mathbf{x} + \mathcal{O}(\|\delta m\|^2). \quad (4.79)$$

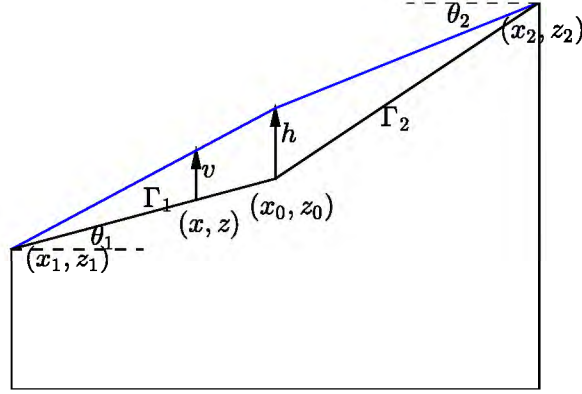


Figure 4.29: Illustration of perturbing vertex: perturbation of vertex along z -axis.

Now, we perturb a vertex along z -axis. For simplicity, we keep our analysis in 2D and our experiments in 2.5D. Figure 4.29 is an illustration for this vertex perturbation. In the figure, we perturb vertex (x_0, z_0) along z -axis to $(x_0, z_0 + h)$. The adjacent vertices of the perturbed vertex are (x_1, z_1) and (x_2, z_2) . Denote the surface connecting (x_1, y_1) and (x_0, y_0) as Γ_1 , and the surface connecting (x_2, y_2) and (x_0, y_0) as Γ_2 . For a given point (x, z) on the moving surface, it moves for distance v , where v is given by

$$v(x) = \begin{cases} \frac{x-x_1}{x_0-x_1} h & \mathbf{x} \in \Gamma_1 \\ \frac{x_2-x}{x_2-x_0} h & \mathbf{x} \in \Gamma_2 \end{cases} \quad (4.80)$$

Following equation (4.79) and parameterizing the boundary curve as $(x, z(x))$, we

can derive the impact of this vertex perturbation as

$$\begin{aligned}
\mathcal{J}[m + \delta m] - \mathcal{J}[m] &\approx \int_{\Omega} \delta m(\mathbf{x}) I(\mathbf{x}) d\mathbf{x} \\
&= (m_1 - m_2) \int_{x_1}^{x_2} \int_{z(x)}^{z(x)+v(x)} I(\mathbf{x}) dz dx \\
&= (m_1 - m_2) \int_{x_1}^{x_2} v(x) I(x, \xi) dx, \quad \text{where } \xi \in (z(x), z(x) + v(x)) \\
&= (m_1 - m_2) \left[\int_{x_1}^{x_0} v(x) I(x, \xi) dx + \int_{x_0}^{x_2} v(x) I(x, \xi) dx \right] \\
&= (m_1 - m_2) \left[\int_{\Gamma_1} v(x) I(x, \xi) \cos \theta_1 d\tau + \int_{\Gamma_2} v(x) I(x, \xi) \cos \theta_2 d\tau \right] \\
&= (m_1 - m_2) \int_{\Gamma_1} \frac{x - x_1}{x_0 - x_1} h I(x, \xi) \cos \theta_1 d\tau \\
&\quad + (m_1 - m_2) \int_{\Gamma_2} \frac{x_2 - x}{x_2 - x_0} h I(x, \xi) \cos \theta_2 d\tau,
\end{aligned} \tag{4.81}$$

where θ_1 and θ_2 are the angles between surface Γ_1, Γ_2 and the x -axis respectively, and they are given by

$$\begin{aligned}
\cos \theta_1 &= \frac{x_0 - x_1}{\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}} \\
\cos \theta_2 &= \frac{x_2 - x_0}{\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}}
\end{aligned} \tag{4.82}$$

Again, letting $h \rightarrow 0$ ($\xi \rightarrow z(x)$) leads to

$$\frac{\partial \mathcal{J}}{\partial z_0} = (m_1 - m_2) \cos \theta_1 \int_{\Gamma_1} \frac{x - x_1}{x_0 - x_1} I(\mathbf{x}) d\tau + (m_1 - m_2) \cos \theta_2 \int_{\Gamma_2} \frac{x_2 - x}{x_2 - x_0} I(\mathbf{x}) d\tau. \tag{4.83}$$

The computational procedure of obtaining this derivative is summarized in algo-

rithm 13.

Algorithm 13: Compute derivative w.r.t. vertex perturbation

Input: a velocity model m , vertex (x_0, z_0) to be perturbed, and surfaces Γ_1, Γ_2 connected to this vertex

Output: derivative of misfit function w.r.t. vertex perturbation $\partial\mathcal{J}/\partial z_0$

- 1 Apply the adjoint-state method (i.e. RTM) to get the image $I(\mathbf{x})$ of the current model m .
 - 2 Identify the set of element surfaces which are subsets of Γ_1 and Γ_2 .
 - 3 Set $\partial\mathcal{J}/\partial z_0 = 0$.
 - 4 **for** each face $\partial D^k \subset \Gamma_1 \cup \Gamma_2$ **do**
 - 5 Add the contribution of the surface integration to $\partial\mathcal{J}/\partial z_0$ by

$$\frac{\partial\mathcal{J}}{\partial z_0} += (m_1 - m_2) \int_{\partial D^k} v(x) \frac{I^+(\mathbf{x}) + I^-(\mathbf{x})}{2} d\tau.$$
 - 6 **Return** $\partial\mathcal{J}/\partial z_0$.
-

4.5.2.1 Multi-vertex example

We now solve an inverse problem where multiple vertices need to be inverted. The gradient is computed by the surface integration approach (algorithm 13). In this experiment, a polyhedral inclusion is located in the bottom of the domain, and the top four vertices of the inclusion are inverted.

The configuration is: the computational domain is a cube of size $[0 \text{ km}, 2 \text{ km}] \times [0 \text{ km}, 0.3 \text{ km}] \times [0 \text{ km}, 2 \text{ km}]$; the velocity is 1.3 km/s inside the inclusion and 1.0 km/s outside the inclusion; three source signals are injected on plane $z = 0.2 \text{ km}$, which are Ricker pulses of 1.5 Hz peak frequency; 27 receivers are evenly distributed on plane $z = 0.35 \text{ km}$; recording end time is $T = 4.0 \text{ s}$. Figure 4.30 also presents the configuration of this experiment. Figure 4.31 is the tetrahedral mesh for the true

velocity model.

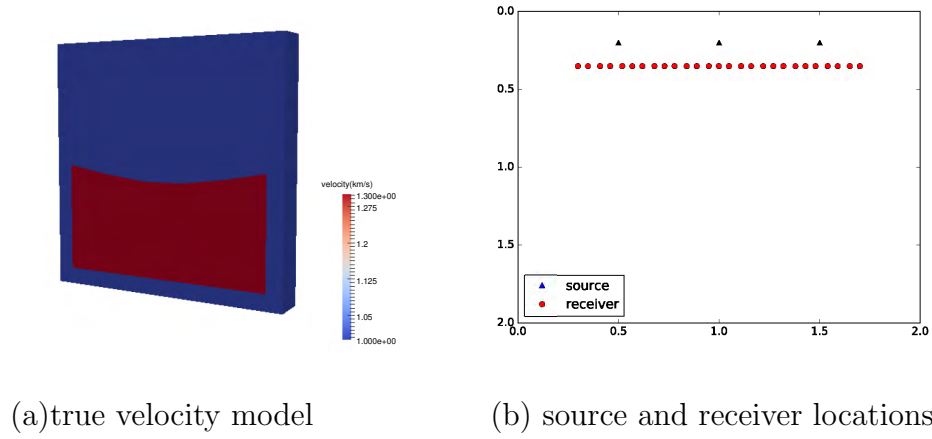


Figure 4.30: Configuration for multi-vertex example: (a) true velocity model, where a ployhedral inclusion is located at the bottom of the domain; (b) the locations of sources and receivers: the blue triangle is the location of the source and the red dots are the location of the receivers.

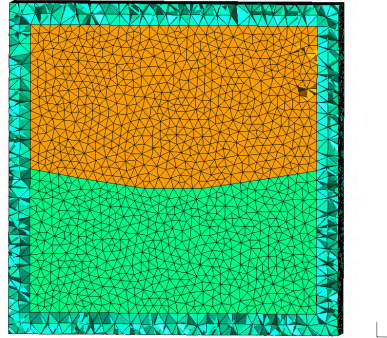


Figure 4.31: Mesh for multi-vertex example.

In figure 4.32, we can see the velocity model has 4 degrees of freedom, where the top 4 vertices are mismatched between the true and initial models. The inverse problem is solved using the surface integration algorithm 13. Figure 4.33 is the result of the FWI experiment, where the solution converges to the true model and the misfit value converges to zero.

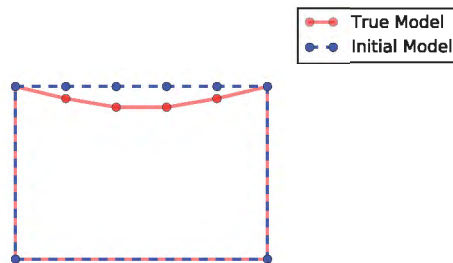


Figure 4.32: True and initial models of the multi-vertex example.

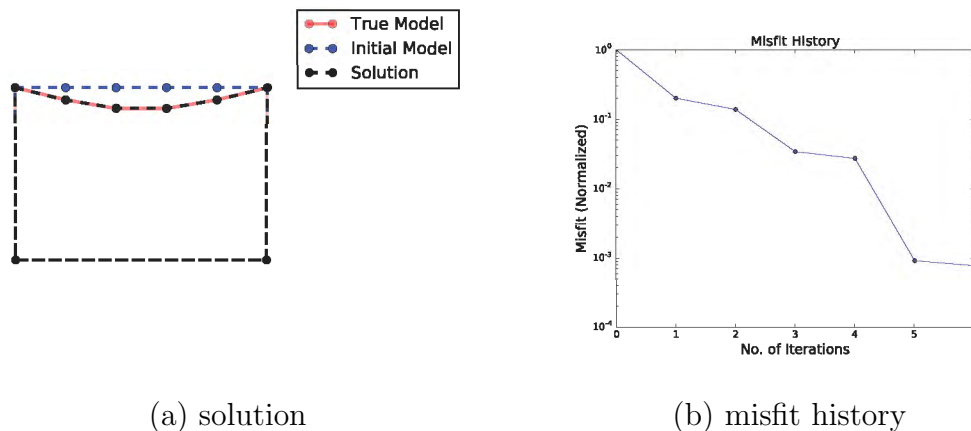


Figure 4.33: Solution of the multi-vertex example: (a) geometry of the solution; (b) misfit history of the FWI iterations. The misfit value drops below 1‰ after 6 iterations.

This multi-vertex example provides the evidence that we can invert many vertices simultaneously. The restriction is: we need to know the direction in advance to perturb any vertices.

4.5.3 Remarks on inverting sharp interfaces

In this section, we have introduced two operations to invert the media interfaces. Assuming the media interfaces are made up of planar surfaces, we can perturb the

planar surfaces along their normal directions and perturb the vertices along a given direction. However, as we do not necessarily know the true normals of the surfaces and the correct directions to move the vertices, we suffer from a strong restriction to apply these operations.

In order to improve this method, more operations must be designed to make it more flexible. For example, we can set criteria to determine when to split a surface and we can introduce operations to rotate a surface. We can use alternative updating to combine many operators together and update surfaces, edges and vertices alternatively in the FWI iterations.

4.6 Summary

In this chapter, we extend the DG-RTM to DG-FWI which is formulated as a PDE-constraint optimization problem. In the DG-FWI implementation, the DG methods solve multiple wave equations, while the RTM subroutine generates the gradient for the optimization problem.

We start the discussion with the derivation of the adjoint-state method. In this chapter, we study the adjoint-state method at both continuous and discrete levels. The adjoint-state method at continuous level leads to an optimize-then-discretize approach, where we first obtain a Fréchet derivative and discretize it without considering the DG discretization. In contrast, the adjoint-state method at the discrete level, known as the discretize-then-optimize approach, takes the DG discretization into consideration in the derivation. As a result, the discrete adjoint-state method can provide a discretely exact gradient while the derivative obtained from continuous adjoint-state method suffers from numerical errors [31]. Unfortunately, we do not exactly follow the discrete adjoint-state method in our implementation due to some computational issues. This prevents us from further studying some gradient-sensitive

methods such as L-BFGS. However, we can still employ steepest descent method and provide the approximated gradient as a descent direction.

We study the conventional FWI with three test cases — a Gaussian inclusion test, a cubic inclusion test and a single layer test. The updated velocity models at each FWI iteration are getting closer to the corresponding true models. However, due to different data coverage and different configurations, the quality of the resulting models varies from case to case.

To highlight the strengths of DGTD, which has smaller interface errors than FDTD, we invert the sharp media interfaces at the end of this chapter. Assuming the interface is made up of planar surfaces, we specify certain perturbations of the interfaces such as moving a surface along its normal direction or moving a vertex along a given direction. In addition, meshes are regenerated to align with the media interfaces at each FWI iteration. With the restriction of the perturbation type, we successfully invert sharp interfaces on simple geometries such as polyhedral inclusions.

Summary and future work

Seismic imaging is a subject of exploration geophysics, and it investigates the subsurface structures. In seismic imaging, algorithms such as reverse time migration (RTM) and full waveform inversion (FWI) require efficient numerical solvers for wave equations. Various numerical methods, such as finite difference methods [111], finite element methods [60], pseudo-spectral methods [19] and discontinuous Galerkin methods [26], have been studied for wave problems. In this dissertation, I exploit the discontinuous Galerkin methods on hybrid meshes for the acoustic wave equation. In particular, my work contributes the following: (1) a discontinuous Galerkin solver on hybrid meshes containing hexahedra, tetrahedra, prisms, and pyramids are developed and accelerated with multi-GPU implementations; (2) the DG solver is applied to two seismic imaging problems — reverse time migration and full waveform inversion. The goal of this work is to exam the availability of DG methods in seismic imaging and highlight the strengths of DG methods in terms of efficiency and accuracy.

As a mesh-based method, DG can easily handle elements of different shapes, which encourages us to use hybrid meshes to improve the computational efficiency. In this work, we have studied four types of elements:

- Hexahedral elements have tensor product properties which accelerate the com-

putation. The total volume operation in hexahedra has a computational cost of $\mathcal{O}(N^{d+1})$ compared to $\mathcal{O}(N^{2d})$ for other three element types, where d is the dimension ($d = 3$ in this work), and N is the polynomial degree in the DG method. Numerical experiments suggest that hexahedral elements have the best efficiency in terms of time cost per degree of freedom.

- Tetrahedral elements have geometric flexibility and are mature for most mesh generation algorithms. The affine mapping of the tetrahedral elements results in constant geometric factors and hence reduces computational cost.
- Prismatic elements take advantage of the low-storage curvilinear DG (LSC-DG) algorithm [71] which saves memory storage by spending additional floating point operations.
- Pyramidal elements have singular points in the vertex mapping, which leads to difficulties in quadrature rules. To resolve this issue, we employ orthogonal rational basis [69] in our DG implementation.

Taking the properties of these element types into consideration, we propose to use hex-dominant meshes for wave simulations. The computational efficiency is achieved through filling most of the domain with hexahedral elements while using tetrahedra, prisms, and pyramids to approximate complicated geometries.

To further improve the computational speed, we employ multi-rate time stepping technique and multi-hardware accelerators. The multi-rate time stepping relaxes the global CFL constraint into multiple local constraints, which allows different elements to take different time step sizes. This technique reduces the total number of time stepping iterations and saves computational time. For multi-hardware acceleration, we use MPI+OCCA to parallelize our implementation into three levels: the entire domain is divided into multiple subdomains; the subdomains consist of many elements; each

element has many degrees of freedom. Subdomains, elements and degrees of freedom correspond to MPI processes, GPU blocks and threads respectively. In this thesis, we also present the implementation details of kernel classification, kernel optimization, data structure, message passing strategy, workload assignment and latency-hiding technique. The acceleration of the DG code enables us to run large-scale simulations on multiple hardware accelerators especially GPUs.

To study DG methods in the context of seismic applications, we first extend the DG solver to the reverse time migration. To begin with, we equip the DG solver with absorbing boundaries to truncate the domain of interest and, and we implement source injections to model signals generated by explosives or airguns. More precisely, perfectly matched layers [115] and point source injection using the scattered-total field formulation [120] are employed in this thesis. Due to the memory limitation on GPUs, we adopt the saving-boundary strategy and reproduce the forward wavefield in the backward phase stage [108]. This saving-boundary technique helps us to trade in computational time cost for memory savings. By using an imaging condition of characteristic fields and high order time quadrature scheme [9], we utilize the intermediate variables in DG methods to improve the imaging quality and successfully recover the subsurface structures in the RTM image.

Finally, the full waveform inversion, which is a PDE-constraint optimization problem, is implemented using the DG solver. As suggested in [31], a discretize-then-optimize approach can provide discretely exact derivatives for the PDE constraint optimization. Therefore, we derive the discrete adjoint-state method depending on our specific DG discretization in this thesis. Unfortunately, our implementation does not exactly follow our theoretical derivation, but we are still able to solve the conventional FWI with the steepest descent method. After validating the DG-FWI implementation on several test cases, we continue investigating DG-FWI to invert

sharp interfaces. In this thesis, we specify the perturbation of a media interface and invert it with mesh regeneration in the FWI iterations. Experiments show that simple geometries such as polyhedral inclusion can be inverted using this technique. As DG methods have smaller interface errors than FDTD [26], inverting sharp interfaces would highlight the strengths of DG methods.

In brief, we have studied the DG methods on hybrid meshes with a numerous consideration of performance and explored the DG solver in the seismic imaging context. The advantage of DG methods over other numerical methods is appealing, but our current implementation still has some weakness that can be improved. The future work may include but not limited to the following:

- Non-conforming meshes: although we build our DG solver on hybrid meshes, the elements we want are tetrahedra and hexahedra. Prisms and pyramids are transitional elements, and they consume significant computational cost. Since DG is good at dealing with non-conforming meshes, we can use non-conforming meshes so that triangular and quadrangular faces can be connected. Removal of prisms and pyramids does not only improve the performance of the DG methods but also simplifies mesh generations.
- Adaptivity: one of the many advantages of DG methods is the adaptivity property. We can perform h or p adaptivity in a relatively simple manner on DG methods. The adaptivity leads to the improved local accuracy in the areas of interest.
- Smooth model coefficients: our current implementation uses piece-wise constant representations of velocity and density models, i.e. on each element, the model parameters are constants. This piece-wise constant model enables representation of sharp interfaces. However, the drawback is smooth models can not

be adequately represented. Fortunately, media heterogeneity in a sub-element scale has already been studied in DG methods. For instance, weight-adjusted DG methods (WADG) [57] uses a weight-adjusted inner product to reduce the storage cost. Techniques like WADG should be adopted to resolve smooth model coefficients.

- More operations for the sharp interface perturbations: as discussed in chapter 4, we have specified certain perturbations of the media interfaces. However, this approach suffers from restrictions on the perturbation types. To improve it, we may introduce more types of perturbations on the media interfaces and combine many of these operations together to invert more complicated models.

As more research of the DG methods are conducted and more DG applications in seismic imaging are studied, the strength of DG methods will eventually be realized, and DG will act as a more important role in seismic imaging.

Bibliography

- [1] R. D. Oldham, “The Constitution of the interior of the earth, as revealed by earthquakes,” *Quarterly Journal of the Geological Society*, vol. 62, no. 1-4, pp. 456–475, 1906. 1
- [2] J. F. Claerbout, “Imaging the earth’s interior,” 1985. 1.1
- [3] G. Nolet, *Seismic tomography: with applications in global seismology and exploration geophysics*. Springer Science & Business Media, 2012, vol. 5. 1.1, 1.1, 4
- [4] E. Baysal, D. D. Kosloff, and J. W. Sherwood, “Reverse time migration,” *Geophysics*, vol. 48, no. 11, pp. 1514–1524, 1983. 1.1, 3
- [5] T. Nemeth, C. Wu, and G. T. Schuster, “Least-squares migration of incomplete reflection data,” *Geophysics*, vol. 64, no. 1, pp. 208–221, 1999. 1.1
- [6] J. Virieux and S. Operto, “An overview of full-waveform inversion in exploration geophysics,” *Geophysics*, vol. 74, no. 6, pp. WCC1–WCC26, 2009. 1.1, 1.1, 1.1, 4, 4.1
- [7] J. F. Claerbout, “Toward a unified theory of reflector mapping,” *Geophysics*, vol. 36, no. 3, pp. 467–481, 1971. 1.1
- [8] N. Whitmore *et al.*, “Iterative depth migration by backward time propagation,” in *1983 SEG Annual Meeting*. Society of Exploration Geophysicists, 1983. 1.1, 3
- [9] A. Modave, A. St-Cyr, W. A. Mulder, and T. Warburton, “A nodal discontinuous galerkin method for reverse-time migration on gpu clusters,” *Geophysical Journal International*, vol. 203, no. 2, pp. 1419–1435, 2015. 1.1, 1.1, 1.2, 1.3, 1.3, 3, 3.2, 3.2, 3.3, 3.4, 3.4, 3.6, 5
- [10] A. Fichtner, *Full seismic waveform modelling and inversion*. Springer Science & Business Media, 2010. 1.1, 1.1, 4

- [11] J. C. Strikwerda, *Finite difference schemes and partial differential equations*. SIAM, 2004. 1.1
- [12] W. W. Symes and T. Vdovina, “Interface error analysis for numerical wave propagation,” *Computational Geosciences*, vol. 13, no. 3, pp. 363–371, 2009. 1.1, 1.2
- [13] Z. Alterman and F. Karal, “Propagation of elastic waves in layered media by finite difference methods,” *Bulletin of the Seismological Society of America*, vol. 58, no. 1, pp. 367–398, 1968. 1.1, 4
- [14] D. M. Boore, “Finite difference methods for seismic wave propagation in heterogeneous materials,” *Methods in computational physics*, vol. 11, pp. 1–37, 1972. 1.1, 4
- [15] K. Kelly, R. Ward, S. Treitel, and R. Alford, “Synthetic seismograms: a finite-difference approach,” *Geophysics*, vol. 41, no. 1, pp. 2–27, 1976. 1.1, 4
- [16] J. Hou and W. W. Symes, “An approximate inverse to the extended born modeling operator,” *Geophysics*, vol. 80, no. 6, pp. R331–R349, 2015. 1.1, 3, 4
- [17] M. O. Deville, P. F. Fischer, and E. H. Mund, *High-order methods for incompressible fluid flow*. Cambridge University Press, 2002, vol. 9. 1.1, 2.3, 2.3.1
- [18] D. D. Kosloff and E. Baysal, “Forward modeling by a fourier method,” *Geophysics*, vol. 47, no. 10, pp. 1402–1412, 1982. 1.1
- [19] B. Fornberg, “The pseudospectral method: Comparisons with finite differences for the elastic wave equation,” *Geophysics*, vol. 52, no. 4, pp. 483–501, 1987. 1.1, 3, 5
- [20] K. J. Marfurt, “Accuracy of finite-difference and finite-element modeling of the scalar and elastic wave equations,” *Geophysics*, vol. 49, no. 5, pp. 533–549, 1984. 1.1
- [21] F. Moser, L. J. Jacobs, and J. Qu, “Modeling elastic wave propagation in waveguides with the finite element method,” *Ndt & E International*, vol. 32, no. 4, pp. 225–234, 1999. 1.1
- [22] S. Minisini, E. Zhebel, A. Kononov, and W. Mulder, “Efficiency comparison for continuous mass-lumped and discontinuous galerkin finite-elements for 3d wave propagation,” in *74th EAGE Conference and Exhibition incorporating EUROPEC 2012*, 2012. 1.1
- [23] J. Tromp, D. Komattisch, and Q. Liu, “Spectral-element and adjoint methods in seismology,” *Communications in Computational Physics*, vol. 3, no. 1, pp. 1–32, 2008. 1.1

- [24] D. Peter, D. Komatitsch, Y. Luo, R. Martin, N. Le Goff, E. Casarotti, P. Le Locher, F. Magnoni, Q. Liu, C. Blitz *et al.*, “Forward and adjoint simulations of seismic wave propagation on fully unstructured hexahedral meshes,” *Geophysical Journal International*, vol. 186, no. 2, pp. 721–739, 2011. 1.1
- [25] V. Monteiller, S. Chevrot, D. Komatitsch, and Y. Wang, “Three-dimensional full waveform inversion of short-period teleseismic wavefields based upon the sem-dsm hybrid method,” *Geophysical Journal International*, vol. 202, no. 2, pp. 811–827, 2015. 1.1
- [26] X. Wang, “Discontinuous galerkin time domain methods for acoustics and comparison with finite difference time domain methods,” Ph.D. dissertation, Rice University, December 2009. 1.1, 1.2, 4.5, 5
- [27] J.-F. Remacle, J. E. Flaherty, and M. S. Shephard, “An adaptive discontinuous galerkin technique with an orthogonal basis applied to compressible flow problems,” *SIAM review*, vol. 45, no. 1, pp. 53–72, 2003. 1.1
- [28] C. Ober, T. Smith, J. Overfelt, S. Collis, G. Von Winckel, B. Van Bloemen Waanders, N. Downey, S. Mitchell, S. Bond, D. Aldridge *et al.*, “Visco-elastic fwi using discontinuous galerkin,” in *SEG Technical Program Expanded Abstracts 2016*. Society of Exploration Geophysicists, 2016, pp. 1190–1195. 1.1, 4
- [29] M. Dumbser and M. Käser, “An arbitrary high-order discontinuous galerkin method for elastic waves on unstructured meshesii. the three-dimensional isotropic case,” *Geophysical Journal International*, vol. 167, no. 1, pp. 319–336, 2006. 1.1
- [30] J. A. Bramwell, “A discontinuous petrov-galerkin method for seismic tomography problems,” 2013. 1.1
- [31] L. C. Wilcox, G. Stadler, C. Burstedde, and O. Ghattas, “A high-order discontinuous galerkin method for wave propagation through coupled elastic-acoustic media,” *Journal of Computational Physics*, vol. 229, no. 24, pp. 9373–9396, 2010. 1.1, 1.2, 4.6, 5
- [32] P. Lailly, “The seismic inverse problem as a sequence of before stack migrations,” in *Conference on inverse scattering: theory and application*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983, pp. 206–220. 1.1, 4
- [33] A. Tarantola, “Inversion of seismic reflection data in the acoustic approximation,” *Geophysics*, vol. 49, no. 8, pp. 1259–1266, 1984. 1.1, 4
- [34] —, “Inverse problem theory: Methods for data fitting and parameter estimation,” 1987. 1.1, 4

- [35] R. G. Pratt, C. Shin, and G. Hick, “Gauss–newton and full newton methods in frequency–space seismic waveform inversion,” *Geophysical Journal International*, vol. 133, no. 2, pp. 341–362, 1998. 1.1
- [36] J. Nocedal, “Updating quasi-newton matrices with limited storage,” *Mathematics of computation*, vol. 35, no. 151, pp. 773–782, 1980. 1.1
- [37] R.-E. Plessix, “A review of the adjoint-state method for computing the gradient of a functional with geophysical applications,” *Geophysical Journal International*, vol. 167, no. 2, pp. 495–503, 2006. 1.1
- [38] L. C. Wilcox, G. Stadler, T. Bui-Thanh, and O. Ghattas, “Discretely exact derivatives for hyperbolic pde-constrained optimization problems discretized by the discontinuous galerkin method,” *Journal of Scientific Computing*, vol. 63, no. 1, pp. 138–162, 2015. 1.1, 1.2, 1.4, 2.2.1, 4, 4.3, 4.3.2
- [39] J. A. Scales, P. Docherty, and A. Gersztenkorn, “Regularisation of nonlinear inverse problems: imaging the near-surface weathering layer,” *Inverse Problems*, vol. 6, no. 1, p. 115, 1990. 1.1, 4
- [40] Y. Yang, B. Engquist, J. Sun, and B. D. Froese, “Application of optimal transport and the quadratic wasserstein metric to full-waveform inversion,” *arXiv preprint arXiv:1612.05075*, 2016. 1.1, 4
- [41] Z. Xue, N. Alger, S. Fomel *et al.*, “Full-waveform inversion using smoothing kernels,” in *2016 SEG International Exposition and Annual Meeting*. Society of Exploration Geophysicists, 2016. 1.1, 4
- [42] M. J. Woodward, D. Nichols, O. Zdraveva, P. Whitfield, and T. Johns, “A decade of tomography,” *Geophysics*, vol. 73, no. 5, pp. VE5–VE11, 2008. 1.1
- [43] C. Shin and Y. H. Cha, “Waveform inversion in the laplace domain,” *Geophysical Journal International*, vol. 173, no. 3, pp. 922–931, 2008. 1.1
- [44] W. Reed and T. Hill, “Triangular mesh methods for the neutron transport equation,” *Los Alamos Scientific Laboratory, Los Alamos, NM*, 1973. 1.2, 2
- [45] B. Rivière, *Discontinuous Galerkin methods for solving elliptic and parabolic equations: theory and implementation*. Society for Industrial and Applied Mathematics, 2008. 1.2
- [46] R. Gandham, D. Medina, and T. Warburton, “Gpu accelerated discontinuous galerkin methods for shallow water equations,” *arXiv preprint arXiv:1403.1661*, 2014. 1.2, 1.3
- [47] J. S. Hesthaven and T. Warburton, *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007. 1.2, 2, 2.2.1, 2.2.2, 2.2.2, 2.2.2, 2.3.2, 2.4.2, 2.7, 2.8, 2.10

- [48] B. Cockburn, G. E. Karniadakis, and C.-W. Shu, *The development of discontinuous Galerkin methods*. Springer, 2000. 1.2, 2
- [49] A. Klöckner, T. Warburton, J. Bridge, and J. S. Hesthaven, “Nodal discontinuous galerkin methods on graphics processors,” *Journal of Computational Physics*, vol. 228, no. 21, pp. 7863–7882, 2009. 1.2, 1.3, 2.4
- [50] J. S. Hesthaven and T. Warburton, “High-order nodal discontinuous galerkin methods for the maxwell eigenvalue problem,” *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 362, no. 1816, pp. 493–524, 2004. 1.2
- [51] P. Monk and G. R. Richter, “A discontinuous galerkin method for linear symmetric hyperbolic systems in inhomogeneous media,” *Journal of Scientific Computing*, vol. 22, no. 1-3, pp. 443–477, 2005. 1.2
- [52] O. B. Matar, P.-Y. Guerder, Y. Li, B. Vandewoestyne, and K. Van Den Abeele, “A nodal discontinuous galerkin finite element method for nonlinear elastic wave propagation,” *The Journal of the Acoustical Society of America*, vol. 131, no. 5, pp. 3650–3663, 2012. 1.2
- [53] B. Riviere and M. F. Wheeler, “Discontinuous finite element methods for acoustic and elastic wave problems,” *Contemporary Mathematics*, vol. 329, pp. 271–282, 2003. 1.2
- [54] M. J. Grote, A. Schneebeli, and D. Schötzau, “Discontinuous galerkin finite element method for the wave equation,” *SIAM Journal on Numerical Analysis*, vol. 44, no. 6, pp. 2408–2431, 2006. 1.2
- [55] M. Ainsworth, P. Monk, and W. Muniz, “Dispersive and dissipative properties of discontinuous galerkin finite element methods for the second-order wave equation,” *Journal of Scientific Computing*, vol. 27, no. 1-3, pp. 5–40, 2006. 1.2
- [56] J. Chan and T. Warburton, “Gpu-accelerated bernstein-bezier discontinuous galerkin methods for wave problems,” *arXiv preprint arXiv:1512.06025*, 2015. 1.2
- [57] J. Chan, R. J. Hewett, and T. Warburton, “Weight-adjusted discontinuous galerkin methods: wave propagation in heterogeneous media,” *arXiv preprint arXiv:1608.01944*, 2016. 1.2, 5
- [58] J.-F. Remacle, R. Gandham, and T. Warburton, “Gpu accelerated spectral finite elements on all-hex meshes,” *Journal of Computational Physics*, vol. 324, pp. 246–257, 2016. 1.2, 1.3, 2.9.1
- [59] G. Seriani and E. Priolo, “Spectral element method for acoustic wave simulation in heterogeneous media,” *Finite elements in analysis and design*, vol. 16, no. 3, pp. 337–348, 1994. 1.2

- [60] D. Komatitsch and J. Tromp, “Introduction to the spectral element method for three-dimensional seismic wave propagation,” *Geophysical journal international*, vol. 139, no. 3, pp. 806–822, 1999. 1.2, 3, 5
- [61] S. J. Sherwin, T. C. Warburton, and G. E. Karniadakis, “Spectral/hp methods for elliptic problems on hybrid grids,” *Contemporary Mathematics*, vol. 218, pp. 191–216, 1998. 1.2
- [62] T. Warburton, “Spectral/hp methods on polymorphic multidomains: Algorithms and applications,” Ph.D. dissertation, Brown University, November 1999. 1.2
- [63] T. Warburton, I. Lomtev, Y. Du, S. Sherwin, and G. Karniadakis, “Galerkin and discontinuous galerkin spectral/hp methods,” *Computer methods in applied mechanics and engineering*, vol. 175, no. 3, pp. 343–359, 1999. 1.2
- [64] R. Kirby, T. Warburton, I. Lomtev, and G. Karniadakis, “A discontinuous galerkin spectral/hp method on hybrid grids,” *Applied numerical mathematics*, vol. 33, no. 1, pp. 393–405, 2000. 1.2
- [65] G. J. Gassner, F. Lörcher, C.-D. Munz, and J. S. Hesthaven, “Polymorphic nodal elements and their application in discontinuous galerkin methods,” *Journal of Computational Physics*, vol. 228, no. 5, pp. 1573–1590, 2009. 1.2
- [66] G. Bedrosian, “Shape functions and integration formulas for three-dimensional finite element analysis,” *International journal for numerical methods in engineering*, vol. 35, no. 1, pp. 95–108, 1992. 1.2, 2.3.4
- [67] M. Bergot, G. Cohen, and M. Duruflé, “Higher-order finite elements for hybrid meshes using new nodal pyramidal elements,” *Journal of Scientific Computing*, vol. 42, no. 3, pp. 345–381, 2010. 1.2, 2.3.4
- [68] M. Bergot and M. Duruflé, “Higher-order discontinuous galerkin method for pyramidal elements using orthogonal bases,” *Numerical Methods for Partial Differential Equations*, vol. 29, no. 1, pp. 144–169, 2013. 1.2, 2.3.4
- [69] J. Chan and T. Warburton, “Orthogonal bases for vertex-mapped pyramids,” *SIAM Journal on Scientific Computing*, vol. 38, no. 2, pp. A1146–A1170, 2016. 1.2, 2.3.4, 2.10, 5
- [70] T. Warburton, “A low storage curvilinear discontinuous galerkin time-domain method for electromagnetics,” in *2010 URSI International Symposium on Electromagnetic Theory*, 2010. 1.2, 2.3.3
- [71] —, “A low-storage curvilinear discontinuous galerkin method for wave problems,” *SIAM Journal on Scientific Computing*, vol. 35, no. 4, pp. A1987–A2012, 2013. 1.2, 2.2.1, 2.3.3, 2.10, 5

- [72] C. W. Gear and D. Wells, “Multirate linear multistep methods,” *BIT Numerical Mathematics*, vol. 24, no. 4, pp. 484–502, 1984. 1.2, 2.5
- [73] T. Warburton, “Accelerating the discontinuous galerkin time-domain method,” *Oberwolfach Report*, vol. 36, pp. 58–60, 2008. 1.2
- [74] N. Gödel, S. Schomann, T. Warburton, and M. Clemens, “Local timestepping discontinuous galerkin methods for electromagnetic rf field problems,” in *Antennas and Propagation, 2009. EuCAP 2009. 3rd European Conference on*. IEEE, 2009, pp. 2149–2153. 1.2, 2.5
- [75] J. Appleyard, J. Appleyard, M. Wakefield, A. Desitter *et al.*, “Accelerating reservoir simulators using gpu technology,” in *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers, 2011. 1.3
- [76] D. Komatitsch, G. Erlebacher, D. Göddeke, and D. Michéa, “High-order finite-element seismic wave propagation modeling with mpi on a large gpu cluster,” *Journal of computational physics*, vol. 229, no. 20, pp. 7692–7714, 2010. 1.3, 1.3
- [77] C. Chen, Z. Wang, D. Majeti, N. Vrilo, T. Warburton, and V. Sarkar, “Optimization of lattice boltzmann simulation using gpu parallel computing and the application in reservoir characterization,” *SPE Journal*, submitted. 1.3
- [78] N. Lopes and B. Ribeiro, “Gpumlib: An efficient open-source gpu machine learning library,” *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 3, pp. 355–362, 2011. 1.3
- [79] A. Gaikwad and I. M. Toke, “Gpu based sparse grid technique for solving multidimensional options pricing pdes,” in *Proceedings of the 2nd Workshop on High Performance Computational Finance*. ACM, 2009, p. 6. 1.3
- [80] N. Wilt, *The cuda handbook: A comprehensive guide to gpu programming*. Pearson Education, 2013. 1.3
- [81] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa, *Heterogeneous Computing with OpenCL: Revised OpenCL 1*. Newnes, 2012. 1.1, 1.2
- [82] D. S. Medina, A. St-Cyr, and T. Warburton, “Occa: A unified approach to multi-threading languages,” *arXiv preprint arXiv:1403.0968*, 2014. 1.3, 2.6
- [83] N. Gödel, S. Schomann, T. Warburton, and M. Clemens, “Gpu accelerated adams–bashforth multirate discontinuous galerkin fem simulation of high-frequency electromagnetic fields,” *Magnetics, IEEE Transactions on*, vol. 46, no. 8, pp. 2735–2738, 2010. 1.3, 2.5, 6

- [84] D. Mu, P. Chen, and L. Wang, “Accelerating the discontinuous galerkin method for seismic wave propagation simulations using the graphic processing unit (gpu)single-gpu implementation,” *Computers & Geosciences*, vol. 51, pp. 282–292, 2013. 1.3
- [85] A. Klöckner, T. Warburton, and J. S. Hesthaven, “Viscous shock capturing in a time-explicit discontinuous galerkin method,” *Mathematical Modelling of Natural Phenomena*, vol. 6, no. 3, pp. 57–83, 2011. 1.3
- [86] M. Fuhry, A. Giuliani, and L. Krivodonova, “Discontinuous galerkin methods on graphics processing units for nonlinear hyperbolic conservation laws,” *International Journal for Numerical Methods in Fluids*, vol. 76, no. 12, pp. 982–1003, 2014. 1.3
- [87] J. Thibault and I. Senocak, “Cuda implementation of a navier-stokes solver on multi-gpu desktop platforms for incompressible flows,” in *47th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*, 2009, p. 758. 1.3
- [88] J. Chan, Z. Wang, A. Modave, J.-F. Remacle, and T. Warburton, “Gpu-accelerated discontinuous galerkin methods on hybrid meshes,” *arXiv preprint arXiv:1507.02557*, 2015. 1.3, 2.2.1, 2.3, 2.4, 2.4.2, 2.4.4, 2.5, 2.9.1, 2.10
- [89] R. Thakur, P. Balaji, D. Buntinas, D. Goodell, W. Gropp, T. Hoefer, S. Kumar, E. Lusk, and J. L. Träff, “Mpi at exascale.” 1.3
- [90] R. Abdelkhalek, H. Calandra, O. Coulaud, J. Roman, and G. Latu, “Fast seismic modeling and reverse time migration on a gpu cluster,” in *High Performance Computing & Simulation, 2009. HPCS’09. International Conference on.* IEEE, 2009, pp. 36–43. 1.3, 3
- [91] D. Komatitsch, D. Göddeke, G. Erlebacher, and D. Michéa, “Modeling the propagation of elastic waves using spectral elements on a cluster of 192 gpus,” *Computer Science-Research and Development*, vol. 25, no. 1-2, pp. 75–82, 2010. 1.3
- [92] S.-W. Cheng, T. K. Dey, and J. Shewchuk, *Delaunay mesh generation*. CRC Press, 2012. 2.3
- [93] G. Karniadakis and S. Sherwin, *Spectral/hp element methods for computational fluid dynamics*. Oxford University Press, 2013. 2.3.1
- [94] D. R. Wells, “Multirate linear multistep methods for the solution of systems of ordinary differential equations,” Illinois Univ., Urbana (USA). Dept. of Computer Science, Tech. Rep., 1982. 2.5
- [95] C. W. Gear, “Automatic multirate methods for ordinary differential equations,” 1980. 2.5

- [96] A. Stock, "Development and application of a multirate multistep ab method to a discontinuous galerkin method based particle in cell scheme," 2009. 2.5
- [97] S. Schomann, N. Godel, T. Warburton, and M. Clemens, "Local timestepping techniques using taylor expansion for modeling electromagnetic wave propagation with discontinuous galerkin-fem," *IEEE Transactions on Magnetics*, vol. 46, no. 8, pp. 3504–3507, 2010. 2.5
- [98] G. Cohen, X. Ferrieres, and S. Pernet, "A spatial high-order hexahedral discontinuous galerkin method to solve maxwells equations in time domain," *Journal of Computational Physics*, vol. 217, no. 2, pp. 340–363, 2006. 2.5
- [99] N. Gödel, N. Nunn, T. Warburton, and M. Clemens, "Scalability of higher-order discontinuous galerkin fem computations for solving electromagnetic wave propagation problems on gpu clusters," *IEEE Transactions on Magnetics*, vol. 46, no. 8, pp. 3469–3472, 2010. 2.6, 2.10
- [100] G. Karypis and V. Kumar, "MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0," <http://www.cs.umn.edu/~metis>, University of Minnesota, Minneapolis, MN, 2009. 2.6
- [101] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities," *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, pp. 1309–1331, 2009. 2.8.2
- [102] D. A. Kopriva and G. Gassner, "On the quadrature and weak form choices in collocation type discontinuous galerkin spectral element methods," *Journal of Scientific Computing*, vol. 44, no. 2, pp. 136–155, 2010. 2.9.2
- [103] K. Yoon, K. J. Marfurt, and W. Starr, "Challenges in reverse-time migration," in *SEG Technical Program Expanded Abstracts 2004*. Society of Exploration Geophysicists, 2004, pp. 1057–1060. 3
- [104] K. Yoon and K. J. Marfurt, "Reverse-time migration using the poynting vector," *Exploration Geophysics*, vol. 37, no. 1, pp. 102–107, 2006. 3
- [105] F. Liu, G. Zhang, S. A. Morton, and J. P. Leveille, "An effective imaging condition for reverse-time migration using wavefield decomposition," *Geophysics*, vol. 76, no. 1, pp. S29–S39, 2011. 3
- [106] Y. Zhang and J. Sun, "Practical issues of reverse time migration: True amplitude gathers, noise removal and harmonic-source encoding," in *Beijing International Geophysical Conference and Exposition 2009: Beijing 2009 International Geophysical Conference and Exposition, Beijing, China, 24–27 April 2009*. Society of Exploration Geophysicists, 2009, pp. 204–204. 3
- [107] W. W. Symes, "Reverse time migration with optimal checkpointing," *Geophysics*, vol. 72, no. 5, pp. SM213–SM221, 2007. 3, 3.3

- [108] R. G. Clapp, “Reverse time migration: Saving the boundaries,” *Stanford Exploration Project*, p. 137, 2008. 3, 3.3, 5
- [109] —, “Reverse time migration with random boundaries,” in *Seg technical program expanded abstracts 2009*. Society of Exploration Geophysicists, 2009, pp. 2809–2813. 3
- [110] M. Araya-Polo, J. Cabezas, M. Hanzich, M. Pericas, F. Rubio, I. Gelado, M. Shafiq, E. Morancho, N. Navarro, E. Ayguade *et al.*, “Assessing accelerator-based hpc reverse time migration,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 147–162, 2011. 3
- [111] R. M. Weiss and J. Shragge, “Solving 3d anisotropic elastic wave equations on parallel gpu devices,” *Geophysics*, vol. 78, no. 2, pp. F7–F15, 2013. 3, 5
- [112] T. Nemeth, J. Stefani, W. Liu, R. Dimond, O. Pell, and R. Ergas, “An implementation of the acoustic wave equation on fpgas,” in *SEG Technical Program Expanded Abstracts 2008*. Society of Exploration Geophysicists, 2008, pp. 2874–2878. 3
- [113] R. G. Clapp, H. Fu, and O. Lindtjorn, “Selecting the right hardware for reverse time migration,” *The leading edge*, vol. 29, no. 1, pp. 48–58, 2010. 3
- [114] J. Virieux, H. Calandra, and R.-É. Plessix, “A review of the spectral, pseudo-spectral, finite-difference and finite-element modelling techniques for geophysical imaging,” *Geophysical Prospecting*, vol. 59, no. 5, pp. 794–813, 2011. 3
- [115] J.-P. Berenger, “A perfectly matched layer for the absorption of electromagnetic waves,” *Journal of computational physics*, vol. 114, no. 2, pp. 185–200, 1994. 3.1.1, 3.1.1, 5
- [116] A. Modave, “Absorbing layers for wave-like time-dependent problems-design, discretization and optimization,” Ph.D. dissertation, University of Liège, October 2013. 3.1.1, 3.1.1
- [117] J.-P. Berenger, “Three-dimensional perfectly matched layer for the absorption of electromagnetic waves,” *Journal of computational physics*, vol. 127, no. 2, pp. 363–379, 1996. 3.1.1
- [118] F. D. Hastings, J. B. Schneider, and S. L. Broschat, “Application of the perfectly matched layer (pml) absorbing boundary condition to elastic wave propagation,” *The Journal of the Acoustical Society of America*, vol. 100, no. 5, pp. 3061–3069, 1996. 3.1.1
- [119] A. Modave, E. Delhez, and C. Geuzaine, “Optimizing perfectly matched layers in discrete contexts,” *International Journal for Numerical Methods in Engineering*, vol. 99, no. 6, pp. 410–437, 2014. 3.1.1

- [120] M. Potter and J.-P. Bérenger, “A review of the total field/scattered field technique for the fdtd method.” 3.1.2, 3.1.2.2, 5
- [121] A. D. Polyanin and V. E. Nazaikinskii, *Handbook of linear partial differential equations for engineers and scientists*. CRC press, 2015. 3.1.2.1, 3.1.2.1
- [122] D. R. Kincaid and E. W. Cheney, *Numerical analysis: mathematics of scientific computing*. American Mathematical Soc., 2002, vol. 2. 3.4
- [123] A. Brougois, M. Bourget, P. Lailly, M. Poulet, P. Ricarte, and R. Versteeg, “Marmousi, model and data,” in *EAGE Workshop-Practical Aspects of Seismic Data Inversion*, 1990. 3.5
- [124] L. Shapiro and R. Haralick, “Computer and robot vision,” *Reading: Addison-Wesley*, vol. 8, 1992. 3.5.3
- [125] Y. Wang, *Seismic inverse Q filtering*. John Wiley & Sons, 2009. 3.5.3
- [126] O. Gauthier, J. Virieux, and A. Tarantola, “Two-dimensional nonlinear inversion of seismic waveforms: Numerical results,” *Geophysics*, vol. 51, no. 7, pp. 1387–1403, 1986. 4, 4.2
- [127] R. Brossier, S. Operto, and J. Virieux, “Seismic imaging of complex onshore structures by 2d elastic frequency-domain full-waveform inversion,” *Geophysics*, vol. 74, no. 6, pp. WCC105–WCC118, 2009. 4
- [128] D. Vigh, B. Starr, J. Kapoor, and H. Li, “3d full waveform inversion on a gulf of mexico waz data set,” in *SEG Technical Program Expanded Abstracts 2010*. Society of Exploration Geophysicists, 2010, pp. 957–961. 4
- [129] L. Sirgue, “The importance of low frequency and large offset in waveform inversion,” in *68th EAGE Conference and Exhibition incorporating SPE EUROPEC 2006*, 2006. 4
- [130] R.-É. Plessix, “Three-dimensional frequency-domain full-waveform inversion with an iterative solver,” *Geophysics*, vol. 74, no. 6, pp. WCC149–WCC157, 2009. 4
- [131] H. Ben-Hadj-Ali, S. Operto, and J. Virieux, “Velocity model building by 3d frequency-domain, full-waveform inversion of wide-aperture seismic data,” *Geophysics*, vol. 73, no. 5, pp. VE101–VE117, 2008. 4
- [132] J. Mao, R.-S. Wu, and B. Wang, “Multiscale full waveform inversion using gpu,” in *SEG Technical Program Expanded Abstracts 2012*. Society of Exploration Geophysicists, 2012, pp. 1–7. 4

- [133] J. Hou, W. W. Symes *et al.*, “Accelerating extended least squares migration with weighted conjugate gradient iteration,” in *2015 SEG Annual Meeting*. Society of Exploration Geophysicists, 2015. 4
- [134] J. R. Krebs, J. E. Anderson, D. Hinkley, R. Neelamani, S. Lee, A. Baumstein, and M.-D. Lacasse, “Fast full-wavefield seismic inversion using encoded sources,” *Geophysics*, vol. 74, no. 6, pp. WCC177–WCC188, 2009. 4
- [135] L. Demanet, “Waves and imaging class notes,” October 2015. 4.2, 4.2.1, 4.2.2
- [136] J. Hudson and J. Heritage, “The use of the born approximation in seismic scattering problems,” *Geophysical Journal International*, vol. 66, no. 1, pp. 221–240, 1981. 4.2.1, 4.2.1
- [137] A. Bradley, “Pde-constrained optimization and the adjoint method,” 2010. 4.3
- [138] S. Wright and J. Nocedal, “Numerical optimization,” *Springer Science*, vol. 35, pp. 67–68, 1999. 4.3.3
- [139] A. Asnaashari, R. Brossier, S. Garambois, F. Audebert, P. Thore, and J. Virieux, “Regularized seismic full waveform inversion with prior model information,” *Geophysics*, vol. 78, no. 2, pp. R25–R36, 2013. 4.4.1
- [140] C. R. Vogel, *Computational methods for inverse problems*. Siam, 2002, vol. 23. 4.5
- [141] E. Esser, L. Guasch, F. J. Herrmann, and M. Warner, “Constrained waveform inversion for automatic salt flooding,” *The Leading Edge*, vol. 35, no. 3, pp. 235–239, 2016. 4.5
- [142] F. Santosa, “A level-set approach for inverse problems involving obstacles,” *ESAIM: Control, Optimisation and Calculus of Variations*, vol. 1, pp. 17–33, 1996. 4.5
- [143] W. Lewis, B. Starr, D. Vigh *et al.*, “A level set approach to salt geometry inversion in full-waveform inversion,” in *2012 SEG Annual Meeting*. Society of Exploration Geophysicists, 2012. 4.5
- [144] Z. Guo, M. V. de Hoop *et al.*, “Shape optimization and level set method in full waveform inversion with 3d body reconstruction,” in *2013 SEG Annual Meeting*. Society of Exploration Geophysicists, 2013. 4.5
- [145] Z. Guo and M. V. de Hoop, “Shape optimization in full waveform inversion with sparse blocky model representations.” 4.5
- [146] S. Schmidt, M. Schütte, and A. Walther, “On theoretical and numerical aspects of the shape sensitivity analysis for the 3d time-dependent maxwells equations,” 2015. 4.5