ABSTRACT

Title of dissertation: NUMERICAL GEOMETRIC ACOUSTICS

Samuel F. Potter

Doctor of Philosophy, 2021

Dissertation directed by: Prof. Maria K. Cameron

Dept. of Mathematics

Sound propagation in air is accurately described by a small perturbation of the ambient pressure away from a quiescent state. This is the realm of linear acoustics, where the propagation of a time-harmonic wave can be modeled using the Helmholtz equation. When the wavelength is small relative to the size of a scattering obstacle, techniques from geometric optics are applicable. Geometric methods such as raytracing are often used for computational room acoustics simulations in situations where the geometry of the built environment is sufficiently complicated. At the same time, the high-frequency approximation of the Helmholtz equation is described by two partial differential equations: the eikonal equation, whose solution gives the first arrival time of a geometric acoustics/optics wavefront as a field; and a transport equation, the solution of which describes the amplitude of that wavefield. Phenomena related to high-frequency acoustic diffraction are frequently omitted from these models because of their complexity. These phenomena can be modeled using a high-frequency diffraction theory, such as the uniform theory of diffraction. Despite their shortcomings, geometric methods for room acoustics provide a useful trade-off between realism and computational efficiency.

Motivated by the limitations of geometric methods, we approach the problem of geometric acoustics using numerical methods for solving partial differential equations. Our focus is offline sound propagation in a high-frequency regime where directly solving the wave or Helmholtz equations is infeasible. To this end, we conduct a broad-based survey of semi-Lagrangian solvers for the eikonal equation, which make the local ray information of the solution explicit. We develop efficient, first-order solvers for the eikonal equation in 3D, called ordered line integral methods (OLIMs). The OLIMs provide intuition about how to design work-efficient semi-Lagrangian eikonal solvers, but their first order accuracy is not sufficient to compute the amplitude consistently. Motivated by the requirements of sound propagation simulations, we develop higher-order semi-Lagrangian eikonal solvers which we term jet marching methods (JMMs). JMMs augment the efficiency of OLIMs by additionally transporting higher-order derivative information of the eikonal in a causal fashion, which allows for high-order solution of the eikonal equation using compact stencils. We use the information made available locally by our JMMs to use paraxial raytracing to simultaneously solve the transport equation yielding the amplitude. We initially develop a JMM which handles a smoothly varying speed of sound on a regular grid in 2D. Motivated by the requirements of room acoustics applications, we develop a second-order JMM for solving the eikonal equation on a tetrahedron mesh for a constant speed of sound as a special case. As before, we use paraxial raytracing to compute the amplitude. Additionally, we compute multiple arrivals by reinitializing the eikonal equation on reflecting walls and diffracting edges. To compute these scattered fields, we devise algorithms which allow us to apply reflection and diffraction boundary conditions for the eikonal and amplitude. For the amplitude, we construct algorithms that allow us to apply the uniform theory of diffraction in a semi-Lagrangian setting efficiently.

NUMERICAL GEOMETRIC ACOUSTICS

by

Samuel Francis Potter

Dissertation submitted to the faculty of the graduate school of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2021

Advisory committee:

Prof. Maria K. Cameron, *advisor/chair*

Prof. Ramani Duraiswami, *co-advisor*

Prof. P. S. Krishnaprasad, *dean's representative*

Prof. Howard Elman

Prof. Ming Lin

ACKNOWLEDGMENTS

First and foremost, I owe a deep debt of gratitude to Ramani Duraiswami. Before arriving at the University of Maryland, I scanned the list of faculty affiliated with the ECE program and immediately identified Ramani as *the* guy to work with, on account of his spatial audio research and involvement in a proliferation of other computational physics research topics. I also noticed that he was offering a special topics course on the fast multipole method, something that seemed esoteric and impressive, but which I knew nothing about. I excitedly related this information to JoJo—then my girlfriend, now my wife—but said I had better wait a few semesters until I was better prepared for such an arcane topic. She informed me that I was an idiot and that it would be stupid not to enroll. I ended up being one of three students registered—and it was the last time the course was offered! Close call.

Taking Ramani's course advanced me directly to the bleeding edge of research in numerical methods and scientific computing, and I haven't looked back. At the end of the semester, I told Ramani that I wanted to join his lab. He grunted in acknowledgement, got up, and showed me a desk. Over the following months, Ramani steered me towards a dizzying array of topics. Each time I asked Ramani a question about an idea I had, some potential research direction, his reply was usually, "Nail and I looked into that 5/10/15/20 years ago... here are some papers." At the same time, Ramani never hesitated to editorialize and loop me into the politics and drama of scientific research by sharing his extensive historical knowledge, making the new world I had entered come alive, and helping me understand how the sausage was made. In short, I got what I asked for, even if I wasn't always ready for it. Ramani took a big chance on me, and for that I simply cannot repay him.

Along the way, I started working with Masha Cameron. Towards the end of the second semester of her offering of the year-long graduate sequence in scientific

computing, we were studying the radix-2 FFT, but skipped non-power-of-two-sized FFTs. After class, I sidled up to her and said she might find a plot of the runtime of MATLAB's `fft` versus the size of the input vector interesting. She returned next class with a big grin on her face, and displayed a beautiful scatter plot with different prime factors color-coded to highlight the different asymptotic prefactors of `fft`'s $O(N \log N)$ runtime, and thanked me for the idea. When she advertised a summer research project related to the fast marching method, I leapt at the opportunity.

Working with Masha these past few years has been the experience of a lifetime. I signed on for a doctorate because I wanted to be trained as a scientist. Masha allowed me to develop my own ideas and seek out applications for them. She trusted my judgment but was critically engaged, not holding back from telling me my ideas were misguided or that I needed to understand something more deeply. This helped me find my feet and trust my own judgment, making it possible for me to channel my creative energy into scientific research. Throughout the process, I never doubted that I was in capable hands, and I am very grateful for having received so much of Masha's time, energy, thoughtfulness, and kindness. Besides this, having an advisor who also turned out to be a close friend and confidant was pure gravy.

My endeavors were aided and abetted by inspiring mentors and collaborators. Thanks to Howard Elman for being a good friend, for offering me concrete and critical support when I needed it most, for answering an endless stream of technical questions, and for taking me seriously. Thanks to Erwan Mazarico and Norbert Schörghofer for their enthusiasm, and for letting me moonlight as a planetary scientist at NASA and PSI. Thanks to John Snyder and Nikunj Raghuvanshi for allowing me to work on spatial audio codecs for a summer at MSR. Thanks to Prof. Krishna for being a phenomenal teacher, a font of wisdom, and a source of inspiration. Thanks to Ming Lin for being kind and approachable, and for sharing her enthusiasm and excitement with me. Finally, special thanks to Alex Vladimirsky for providing crucial, last minute

support, and for shooting straight with me. I would not have made it to this point without any of your help.

Of course, the person I'm more deeply indebted to than all the rest is JoJo. We left for the east coast to have an adventure, and we've certainly had that. I can't imagine having taken it with someone else, and that the adventure continue is all I ask. Thanks for sticking with me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this dissertation, we develop a variety of semi-Lagrangian solvers for the eikonal equation, and investigate their applicability to solving problems stemming from computational room acoustics:

- First, in this introductory chapter, we present detailed background material that contextualizes our work and is used throughout the remaining chapters. This includes material on high-frequency wave propagation, background on computational room acoustics, numerical methods for solving the eikonal equation (prior art), a short introduction to paraxial ray theory, and boundary conditions for computing multiple arrivals, including the uniform theory of diffraction. We also discuss related work and how our approach compares with the most obvious alternative: raytracing.

- In Chapter 2, we present a broad-based survey on first-order semi-Lagrangian solvers (ordered line integral methods, or OLIMs) for the eikonal equation in 3D, including fast hierarchical algorithms in 3D, and numerical analysis of a solver based on a simplified midpoint rule for integrating the action integral. This work is based on our publication, "Ordered line integral methods for solving the eikonal equation" [101]. These algorithms were also applied to develop OLIMs

for computing the quasipotential in 3D, which was published in "Computing the quasipotential for nongradient SDEs in 3D" [140]. We include an example related to computing multiple arrivals in 2D to motivate the need for higher-order eikonal solvers for room acoustics applications.

- In Chapter 3, we present a fully developed family of second and third order semi-Lagrangian solvers for the eikonal equation on a regular grid in 2D. We conduct extensive numerical experiments showing the trade-offs in our design choices, and present some preliminary theoretical results. We also develop algorithms which use paraxial raytracing locally to march the geometric spreading in order to compute the amplitude for the high-frequency approximation of the Helmholtz equation. This chapter is based on material from a paper that is currently in review. A preprint is available as "Jet marching methods for solving the eikonal equation" [102].

- In Chapter 4, we introduce a JMM for solving the eikonal equation with a constant speed of sound on a tetrahedron mesh conforming to a complicated polyhedral boundary. We introduce a new approach to ensuring causality in this setting so that a Dijkstra-like algorithm can be used, and we develop dynamic programming algorithms for applying the multiple arrival boundary conditions so that we can reinitialize the eikonal equation along reflecting walls and diffracting edges to compute scattered fields. We conduct preliminary numerical tests and examples computing multiple arrivals for a simple building test problem with multiple rooms, columns, and recessed architectural features. This work is in preparation and will be submitted as "Precomputed eikonal-based sound propagation".

- Finally, in Chapter 5, we summarize our results and collect a variety of future research directions in the form of a short prospectus.

1.1 High-frequency acoustics

The simulation of room acoustics begins with the acoustic wave equation:

$$p_{tt} - c^2 \Delta p = f, \tag{1.1}$$

a partial differential equation (PDE), where $p = p(\boldsymbol{x}, t)$ is the sound pressure, c is the speed of sound in air, and f is some unspecified boundary data, which might model a sound source. The speed of sound c is of central importance. It can vary both in time and space for a variety of reasons, but in our simple model we will assume that it is linearly related to temperature—i.e., $c = 331.4 + 0.6T$, where T is the temperature in Celsius [77]. Throughout a built environment, particularly in a large, partially outdoors space, the speed of sound may easily vary by several percent or more. Other effects such as a gradient in wind speed can affect sound propagation. By contrast, for the propagation of seismic waves in the earth, c can vary quickly or even be discontinuous. In this dissertation, we will focus on a slowly varying, smooth speed of sound, including a constant of speed of sound as a special case. We do not consider a time-varying speed of sound at all.

If we assume that the sound pressure is time-harmonic, $p(\boldsymbol{x}, t) = P(\boldsymbol{x})e^{i\omega t}$, for some frequency $\omega > 0$, then P satisfies the Helmholtz equation:

$$\left(\Delta + \frac{\omega^2}{c(\boldsymbol{x})^2}\right) P(\boldsymbol{x}) = 0. \tag{1.2}$$

The wave equation can be reconstructed by using linear superposition to combine particular solutions of (1.2) for varying $\omega > 0$; i.e., by integrating over ω, thereby computing the inverse Fourier transform.

Our particular interest is in doing high-frequency room acoustics simulations. We think of a wave as being "high-frequency" if its wavelength is small relative to an obstacle that it might be scattered from. A rule of thumb is that there should be at least 10 wavelengths to the diameter of the scattering object. Human hearing spans

from 20 Hz to 20 kHz; or, from wavelengths of about 17 m down to 1.7 cm. In the built environment, we find scattering obstacles of all sizes: the individual features of furniture can be quite small, on the order of millimeters or centimeters, while architecture features can be quite large, possessing scattering cross-sections on the order of meters in the diameter. Naturally, buildings themselves constitute still larger scatterers, for which the majority of the audible spectrum can be treated reasonably by the high-frequency approximation.

A high-frequency approximation of the solution of (1.2) can be obtained using the so-called WKB, geometric optics (GO), or geometric acoustics (GA) ansatz [89, 16], in which sound waves are assumed to be accurately modeled by tracing acoustic rays. In this work, we will refer to this as "geometric acoustics". This may be familiar from raytracing in computer graphics, but a key difference here is that the speed of sound possesses a long wavelength, unlike the speed of light, and varies continuously on the same scale as the space which we inhabit. So, we must incorporate both curved rays and diffraction from singular features of the domain. These are phenomena familiar from everyday life: if we yell into the wind, our speech becomes unintelligible due to it bending upward (although this is not due principally to varying temperature); and, although we can easily hear around corners, we cannot see around them. The former phenomenon is explained by a continuously varying speed of sound, while the latter is explained by the macroscopic wavelength of sound. There are still other interesting acoustic phenomena which we encounter regularly. Sound waves can easily transmit through solid materials, a car horn pitches up and down as it drives past us, etc. While raytracing is a useful means of modeling sound propagation quickly, it leaves out some of these phenomena, preventing it from being completely appropriate for simulating sound propagation. There are modifications which can be made, but the process becomes more computationally intensive. The contents of this thesis aim to advance the state of art of GA methods for the simulation of sound propagation

which do not explicitly involve the tracing of rays.

The GA ansatz goes as follows. We let $\omega \gg 0$ be a large frequency parameter, and assume that:

$$P(\boldsymbol{x}) \sim \alpha(\boldsymbol{x}) e^{i\omega\tau(\boldsymbol{x})} \tag{1.3}$$

Here, we call α the amplitude, and τ the eikonal. By substituting (1.3) into eq. (1.2) and collecting like powers, we obtain the eikonal equation:

$$\|\nabla\tau\| = \frac{1}{c} = s, \qquad s \equiv \frac{1}{c}, \qquad \boldsymbol{x} \in \Omega, \tag{1.4}$$

where we have defined $s = s(\boldsymbol{x})$, the slowness, which is often more convenient than c for calculations. We also obtain a transport equation for α which depends on the solution of (1.4):

$$\alpha\Delta\tau + 2\nabla\alpha \cdot \nabla\tau = 0. \tag{1.5}$$

Note that if τ and α together satisfy (1.4) and (1.5), then the accuracy of the GA ansatz is $O(\omega^{-1})$:

$$P(\boldsymbol{x}) = \alpha(\boldsymbol{x}) e^{i\omega\tau(\boldsymbol{x})} + O(\omega^{-1}). \tag{1.6}$$

It is also possible to develop a series GA ansatz, which again results in the eikonal equation, but this time attached to an infinite sequence of amplitude terms, which each provide a progressively higher-order correction:

$$P(\boldsymbol{x}) \sim e^{i\omega\tau(\boldsymbol{x})} \sum_{m=0}^{\infty} \frac{\alpha_m(\boldsymbol{x})}{(-i\omega)^m}. \tag{1.7}$$

Here, τ again satisfies (1.4), and α_0 satisfies (1.5). For $m > 0$, we have that α_m satisfies:

$$\alpha_m(\boldsymbol{x})\nabla\tau(\boldsymbol{x}) + 2\nabla\tau(\boldsymbol{x}) \cdot \nabla\alpha_m(\boldsymbol{x}) = \Delta\alpha_{m-1}(\boldsymbol{x}). \tag{1.8}$$

This is an asymptotic sequence and may not converge [97]. We will not dwell on this in this work, since we limit ourselves to exploring the effectiveness of using the single term high-frequency approximation of (1.6), but it is an important avenue for future research.

Equations (1.4) and (1.5) are PDEs and must be supplied with meaningful boundary conditions. We write:

$$\begin{aligned} \|\nabla\tau(\boldsymbol{x})\| &= s(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega, \\ \tau(\boldsymbol{x}) &= g(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Gamma, \end{aligned} \tag{1.9}$$

for the eikonal equation. Boundary conditions for the amplitude are less straightforward, and require matching short- and long-range asymptotics. To avoid dealing with this directly, we take advantage of the fact that the speed of sound will be a constant plus a small perturbation. Then, in a small neighborhood of a point source, we assume that the speed of sound is constant. This lets us model the pressure as a monopole:

$$P(\boldsymbol{x}) \sim \frac{e^{-ik\|\boldsymbol{x}-\boldsymbol{x}_{\text{src}}\|}}{\|\boldsymbol{x}-\boldsymbol{x}_{\text{src}}\|} = \alpha(\boldsymbol{x})e^{i\omega\tau(\boldsymbol{x})}, \tag{1.10}$$

where $\boldsymbol{x}_{\text{src}} \in \Omega$ is the position of the point source, so that $\alpha(\boldsymbol{x}) \sim \|\boldsymbol{x}-\boldsymbol{x}_{\text{src}}\|^{-1}$ near $\boldsymbol{x}_{\text{src}}$ [112].

What is the physical significance of τ and α? The eikonal, τ, has units of time, and describes the level sets of an expanding wavefront. The amplitude has units of pressure, and describes the decay of the amplitude of the wavefront as it expands. Both τ and α are—for the most part—smooth, and slowly varying quantities. There are special exceptions to this: at rarefaction fans and shock lines for the eikonal, and at caustics for the amplitude. But the fact that these functions are smooth almost everywhere constitutes their main advantage over direct solution of eq. (1.2). Namely, at high frequencies, P oscillates rapidly, which drives up the computational cost of a Helmholtz solver. At the same time, the eikonal equation admits a type of weak solution called the viscosity solution, which is the natural solution to seek when viewing the eikonal equation from the perspective of dynamic programming or control theory [39]. In wave propagation, it corresponds to the first-arrival time of a GO/GA wave. We will view the viscosity solution as a useful tool which supports developing numerical methods that allow for global solutions to be obtained using

simple dynamic programming algorithms. This provides what we believe to be the primary advantage of our approach over raytracing.

The foregoing discussion neglects a key detail which by now may seem obvious. These two fields, the solutions of (1.4) and (1.5), describe the first arrival of an acoustic wave. Leaving aside the validity of our high-frequency approximation, it is clear that sound in real life is comprised of many distinct, sometimes diffuse, wavefronts, propagating throughout space at all times! How to account for this omission? The reality is that an asymptotic approximation of P is better written as:

$$P(\boldsymbol{x}) \sim \sum_{\substack{k\text{th ray} \\ \text{through } \boldsymbol{x}}} A^{(k)}(\boldsymbol{x}) e^{i\omega\tau^{(k)}(\boldsymbol{x})}. \tag{1.11}$$

There are many rays passing through each point $\boldsymbol{x}$, with various amplitudes and travel times given by $\tau^{(k)}$ and $A^{(k)}$, $k = 0, 1, \ldots$. These rays cannot even be totally ordered, since we may have two distinct rays arriving at a point simultaneously so that $\tau^{(k)}(\boldsymbol{x}) = \tau^{(k')}(\boldsymbol{x})$, for $k \neq k'$.

The terms of eq. (1.11) come about as a result of the sound field propagating throughout the domain Ω, generating rays reflecting and diffracting from obstacles, as well as from caustics. If we consider an expanding wavefront (say, emanating initially from a point source), we can imagine it enveloping a smooth surface which is a subset of $\partial\Omega$, and reflecting therefrom. Likewise for diffraction from a singular subset of $\partial\Omega$—in our model, we augment GA with a high-frequency diffraction theory referred to as the geometric theory of diffraction (GTD) which predicts diffracted rays produced by a wavefront enveloping a singular part of $\partial\Omega$. And there are caustics: subsets of Ω where rays focus to a point, resulting in a region filled with multiple arrivals. We consider these different types of overlapping wavefronts to be separate discrete wavefronts.

To define these discrete wavefronts more precisely, we need to elaborate on what

is meant by a viscosity solution. A solution formula is given by:

$$\tau(\boldsymbol{x}) = \min_{\substack{\boldsymbol{y}\in\Omega \\ \psi:[0,1]\to\Omega \\ \psi(0)=\boldsymbol{y},\psi(1)=\boldsymbol{x}}} \left\{ \tau(\boldsymbol{y}) + \int_0^1 s(\boldsymbol{y}) \|\psi'(\sigma)\| d\sigma \right\}. \tag{1.12}$$

Here, ψ is a ray leading from $\boldsymbol{y}$ to $\boldsymbol{x}$. This is Fermat's principle, stating that the first-arrival ray leading from $\boldsymbol{y}$ to $\boldsymbol{x}$ is the one that takes the least time; indeed, ψ is the extremal minimizing the Fermat functional, (1.12). If Ω is nonconvex, which will usually be the case in interesting room acoustics applications, then (1.12) indicates that the solution of (1.4) will naturally diffract around obstacles, and numerical methods for its solution exploit this fact. This is one of the beauties of the eikonal equation as a modeling tool, and is of critical importance in many applications, particularly those related to optimal control, robotics, and the like. However, we will also be interested in being able to tell which points have been reached by a ray *that has not undergone diffraction*. We will still use numerical methods which compute the full viscosity solution, including diffraction. This extra information will prove invaluable and will help form the scaffolding on which our algorithms are built.

Since the eikonal is continuous across successive "scattering events", taken together, the eikonal fields for each discrete wavefront corresponding to the terms of (1.11) comprise the branches of a multivalued function which we term the multipath eikonal. Each branch will have an associated amplitude. Since the branches corresponding to later arrivals depend on data derived from earlier arrivals, it is clear that the discrete wavefronts can be computed recursively. Hence, for given boundary data, our overarching goal is to compute the multipath eikonal by recursively propagating discrete wavefronts, patching them together using matched boundary conditions derived from the governing equations.

1.2 Precomputed room acoustics

The previous section outlined the basic physical model, and indicated the problem that we would like to develop numerical methods to solve. In this section, we consider why we might want to do this. We are interested in computing the room impulse response (RIR) for a pair of points in Ω. To do this, we place a point source at $\boldsymbol{x}_{\text{src}} \in \Omega$, setting $f(\boldsymbol{x}, t) = \delta(\boldsymbol{x} - \boldsymbol{x}_{\text{src}}, t)$ in (1.1). Hence, $F(\boldsymbol{x}) = \delta(\boldsymbol{x} - \boldsymbol{x}_{\text{src}})$ in (1.2), and $g(\boldsymbol{x}_{\text{src}}) = 0$ in (1.4). The boundary conditions for (1.5) are more complicated. Our goal is to compute the mulipath eikonal in order to recover some of the information contained in the Green's function for the acoustic wave equation, and then use this information to do room acoustics simulations for use in interactive multimedia, such as virtual reality, games, etc.

If we have a numerical method for computing the multipath eikonal for a particular $\boldsymbol{x}_{\text{src}}$ along with associated amplitude field, then we can recover the sequence of arrivals $(\tau^{(k)}, \alpha^{(k)})$ at a target point $\boldsymbol{x} \in \Omega$. Since amplitude generally decreases with successive arrivals, we may be interested in computing all arrivals such that $|\alpha^{(k)}|$ is above a certain threshold, or the first K arrivals, i.e. k such that $0 \leq k < K$. Even using an efficient algorithm, precomputing all of this information is too costly to do at runtime. Instead, we will try to precompute this information so that it can be looked up quickly—for instance, as a user moves around a virtual environment.

If a discretization of Ω consists of N elements, and if we computed the first K arrivals for each source location $\boldsymbol{x}_{\text{src}}$, then the resulting data structure consisting of the "baked" acoustics would require roughly $O(KN^2)$ memory. There are ways of reducing these memory requirements. If we compute a high-order accurate approximation of τ, then we can reduce N. At the same time, if we know that a user is constrained so that their head stays roughly within a 2D surface spanning the navigable area of the domain (a "navigation mesh"), then we can take advantage of Helmholtz reciprocity to simulate "point listeners" instead of point sources. This can make sense, since in

interactive media sound sources are more likely to be placed arbitrarily than listeners. With this restriction, the number of choices of $\boldsymbol{x}_{\text{src}}$ drops from $O(N)$ to $O(N^{2/3})$ so that the overall cost is $O(KN^{5/3})$.

In general, our approach should be effective at recovering the early arrivals quickly and with a reasonable degree of accuracy. A key difference is that our solvers directly compute the directional of arrival, while for other solvers it takes extra effort in a postprocessing step to do so. In other approaches based on solving (1.1) and encoding the result using parameters based on salient psychoacoustic features, the features associated with early arrival information are, generally-speaking, geometric acoustics quantities (e.g., the first-arrival time, the first-arrival loudness, etc.). One of the central hypotheses of this work is that it is best to use high-frequency methods to compute high-frequency quantities, and that we can use Eulerian (really, semi-Lagrangian) techniques to do so globally.

1.3 Numerical methods for solving the eikonal equation

An historical overview of eikonal solvers. There is a vast literature on numerical methods for solving the eikonal equation. The most well-known is Sethian's fast marching method [116]. The fast marching method was developed independently by Tsitsiklis [134]. The two methods can be shown to be equivalent [120]. The FMM was originally based on a first-order accurate upwinded finite difference discretization of the eikonal equation, but was later extended to use higher-order stencils, resulting in (at least formally) a second-order solver [118]. Another formally second-order gradient-augmented marcher based on finite differences was developed for unstructured meshes [119]. Marchers for solving the eikonal equation on unstructured meshes thus far have focused on enforcing monotone causality by using complicated splitting

sections or unfolding the mesh to flatten it locally [88, 72, 119]. Higher-order FMMs based on finite differences have been developed [3].

Another approach to solving a discretized version of eq. 1.4 is the fast sweeping method [132, 142, 139, 82, 143]. Unlike Dijkstra-like methods, which are direct solvers, the fast sweeping method is an iterative solver using an upwind scheme and rotating sweep directions, which obtains $O(N)$ complexity—however, the constant in this asymptotic estimate depends heavily on how frequently the characteristics change direction, and how complicated the geometry of the domain is. Fast sweeping methods have been extended to Hamilton-Jacobi equations [132, 68], and hybrid methods combining the fast sweeping method with a Dijkstra-like method have been introduced recently [29, 30]. Additionally, higher-order fast sweeping methods based on ENO and WENO schemes have been developed [142]. Luo and Zhao also provide a detailed investigation of the convergence properties of fast sweeping methods for static convex Hamilton-Jacobi equations, but with a focus on the 2D case [83].

The aforementioned higher-order solvers based on finite difference stencils require wide stencils and a regular grid. Tsitsiklis's algorithm is a semi-Lagrangian method, where finite differences are replaced by local variational problems obtained by discretizing Fermat's principle [134]. First-order semi-Lagrangian schemes on unstructured meshes have been developed [22, 67]. At the same time, there are high-order Eulerian marchers based on compact finite difference stencils [11].

A quick introduction to solving the eikonal equation. The standard method for solving the eikonal equation is the fast marching method (FMM) [116]. The key feature of the FMM is that it propagates the solution throughout the domain in one pass, from smaller values to larger values, without iteration—hence, it is a direct solver. Indeed, the organizing idea behind the FMM is to use Dijkstra's algorithm combined with upwinded finite differences to solve (1.4) in optimal time. If N is the

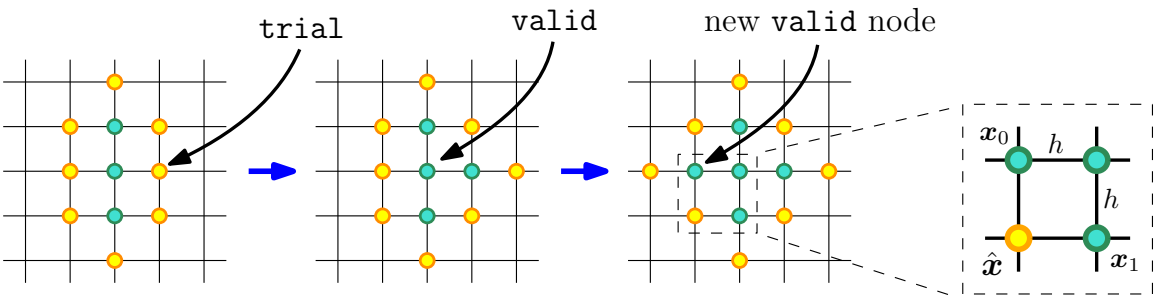


Figure 1.1: Several steps of the fast marching method. Nodes which have a `valid` state are shown as green circles, those with a `trial` state as yellow circles, and `far` nodes are left unmarked.

number of grid points, then the runtime of the FMM is $O(N \log N)$. Three states are introduced: `far`, `trial`, and `valid`. A priority queue is used to order the nodes $\boldsymbol{x}_{i,j}$ with a `trial` state in increasing order of $T(\boldsymbol{x}_{i,j})$. Initially, nodes with boundary values are inserted into the priority queue with a `trial` state, and all other nodes have their state initially set to `far`. Then, nodes are removed one at a time from the priority queue. Each time a node is removed, its state is set to `valid`, its `far` neighbors have their state set to `trial`, and the value of T for each of these `trial` nodes is updated. To update a node $\boldsymbol{x}_{i,j}$, we scan for `valid` neighbors, and solve (1.13) to obtain a new value for $T(\boldsymbol{x}_{i,j})$, where we keep the minimum value of $T(\boldsymbol{x}_{i,j})$ so obtained; afterwards, the position of $\boldsymbol{x}_{i,j}$ is adjusted. Note that a node can only have its priority increased, since the value of $T(\boldsymbol{x}_{i,j})$ will never be increased.

The FMM discretizes (1.4) using first-order one-sided Taylor approximations. For example, on a regular grid in 2D with a grid spacing $h > 0$, and nodes indexed as $\boldsymbol{x}_{i,j}$, we get:

$$\big(T(\boldsymbol{x}_{i,j}) - T(\boldsymbol{x}_{i-1,j})\big)^2 + \big(T(\boldsymbol{x}_{i,j}) - T(\boldsymbol{x}_{i,j-1})\big)^2 = h^2 s(\boldsymbol{x}_{i,j})^2, \tag{1.13}$$

where T denotes a numerical approximation of the eikonal. This yields one of four finite difference stencils surrounding $\boldsymbol{x}_{i,j}$, with the others easily enumerated. If we assume that $T(\boldsymbol{x}_{i,j-1})$ and $T(\boldsymbol{x}_{i-1,j})$ are fixed, then (1.13) is a quadratic equation in $T(\boldsymbol{x}_{i,j})$, whose solution poses no difficulties. A new value of $T(\boldsymbol{x}_{i,j})$ can be obtained

by solving the quadratic equation for each stencil (1.13) where the neighbors $\boldsymbol{x}_{i\pm1,j}$ and $\boldsymbol{x}_{i,j\pm1}$ are both `valid`, and taking the minimum. The resulting characteristic passes through the interval $[\boldsymbol{x}_{i\pm1,j}, \boldsymbol{x}_{i,j\pm1}]$, where $\boldsymbol{x}_{i\pm1,j}$ and $\boldsymbol{x}_{i,j\pm1}$ are the neighboring nodes for the minimizing update. See Figure 1.1 for a pictorial depiction of several steps of the FMM.

There are several questions we can ask about this algorithm. Is $O(N \log N)$ optimal? Does the method converge, and what is its order of accuracy? Are there higher-order methods?

First, the time complexity $O(N \log N)$ is essentially optimal. Another method for solving the eikonal equation is the so-called fast sweeping method (FSM) [143]. The basic version of this method works by using the same upwinded discretization given by (1.13), but replaces the Dijkstra-like approach of the FMM with rotating Gauss-Seidel sweeps. In simple domains, this method computes the solution in a finite number of sweeps, requiring $O(N)$ time total. However, the method has a constant that depends on the geometry of Ω. For example, if we consider a spiral maze with $O(N)$ rotations, as we increase N, we will require $O(N)$ sweeps, resulting in an $O(N^2)$ algorithm. This is a bit of an unfair comparision, but if we imagine rectifying this problem by avoiding unnecessary solves in each sweep, we are led back to the dynamic programming formulation of the FMM. Dynamic programming feels like the correct approach for solving the eikonal equation on a complicated, nonconvex domain.

Another approach is to replace Dijkstra's algorithm with Dial's algorithm [43]. That this was a viable approach was pointed out by Tsitsiklis, who noted that using upwinded finite differences derived from a node's four nearest neighbors (or six, in 3D), was incompatible with this approach, and that instead it was necessary to use eight [134]. The picture is more complicated in 3D [101]. The reason for this has to do with the causality of the solver. The idea behind Dial's algorithm is to simultaneously relax all nodes neighboring the `valid` front whose tentative values are only a small

constant (given by the update gap) greater than their `valid` neighbors. If we imagine a point source at the center of a box with constant speed of sound c, then at each step of Dial's algorithm, we will be able to relax a large fraction of the nodes on an expanding spherical front; this intuition suggests that in 3D we can relax $O(N^{2/3})$ nodes at each step, resulting in $O(N^{1/3})$ steps overall. This allows us to replace the priority queue with a "bucketed" priority queue; or, rather, replace a heap sort with a bucket sort. This simplifies the algorithm significantly, and reduces the total cost to $O(N)$.

In either case, if we are a little more precise about the cost of the FMM, we can break it down into two parts: the time spent on updating nodes, and the time spent on managing the priority queue. Updating nodes is a fixed $O(N)$ cost for the FMM. Since nodes transition from `far` to `trial` to `valid`, without going back and forth between states, we know that the total number of updates must be $O(N)$; and, for the FMM, the cost of a single update is $O(1)$, since it just involves solving a quadratic equation. On the other hand, managing the priority queue is easy and efficient, since this can be done using an array-based binary heap; the cost of which is easily dominated by the cost doing updates, especially considering that the number of nodes in the priority queue at any point in time is $O(N^{2/3})$. Since we will be particularly interested in replacing the update scheme with approaches that trade speed for accuracy, the cost of managing the priority queue quickly fades into the background and can be safely ignored. The fact that a Dijkstra-like "only" runs in $O(N \log N)$ time is *not* a good reason to seek $O(N)$ alternatives. Our view is that these alternatives must also possess some other demonstrable advantage over Dijkstra-like algorithms in order to justify them.

One such advantage may be parallelism. The fast sweeping method can be parallelized using well-known domain decomposition strategies [144, 42]. On the other hand, we have found parallelizing Dial-like algorithms to be an intriguing yet difficult

parallel programming challenge.[1] A Dial-like algorithm should be easier to vectorize than a Dijkstra-like algorithm, by dint of the fact that more than one node can be relaxed simultaneously, so individual updates can be vectorized. Given that modern CPUs make heavy use of vectorization, and since Dijkstra-like algorithms typically exhibit almost no vectorization whatsoever, even a serial implementation of Dial's algorithm has the potential to increase the throughput of floating point operations in the solution of the eikonal equation by an order of magnitude. This cannot be overemphasized, and to date, this problem does not appear to have been seriously investigated. We defer this problem for future work.

Second, does the method converge? The answer is: yes, but the order of convergence can be tricky to pin down. Formally, the FMM converges with $O(h)$ accuracy. However, if the boundary data consists of a point source, then the accuracy degrades to $O(h \log \frac{1}{h})$ [143]. If we consider the behavior of this reduction in accuracy, it relates to a large curvature near a point source singularity, which cannot be accurately captured by the linear interpolation of the first-order finite difference stencils used by the basic FMM or FSM. As the mesh refines, the singularity in $\nabla\tau(\boldsymbol{x}_{\text{src}})$ outpaces the accuracy of (1.13), and the order of convergence degrades. This can be resolved either by initializing with $O(h)$ accuracy, or by solving a factored eikonal equation in a constant radius ball surrounding $\boldsymbol{x}_{\text{src}}$ [52]. There are complications with either approach, but they are widely used, and help obtain the full $O(h)$ accuracy of the FMM.

Unfortunately, point sources are not the only place where $\nabla\tau$ can develop a singularity which degrades the order of accuracy. It is easy to see that if the eikonal diffracts around an obstacle, there will be a shadow boundary separating the direct and diffracted parts of the solution, and that τ is only C^1 across that boundary. This discontinuity in $\nabla\tau$ is generated by a rarefaction fan that is created at the edge of

[1]We would like to thank Manyuan Tao for her extensive help parallelizing Dial-based OLIMs and JMMs in 2D and 3D.

the sharp, diffracting obstacle. In 2D, a rarefaction fan behaves like a point source, and a line source in 3D. So, it becomes necessary not only to ensure a high order of accuracy downwind from a point source, but also all diffracting features. This has been explored in 2D, but has been uninvestigated as yet in 3D [106].

From the foregoing discussion, we can see that order of convergence, diffraction, factoring, grid shape, stencil size, boundary conformance... all these come together to influence one another, and cannot be considered separately. The fundamental problem is essentially a geometric one: in the regions where τ is C^∞, convergence should be rapid and easily obtained, but this hinges on how accurately the subsets of Ω across which τ is C^1. This includes the physical boundary of the domain, $\partial\Omega$, as well as the subsets intrinsic to τ: the shadow boundary, the point source location, etc. Numerically, this requires accurate meshing and conforming elements. So, although we can try to shake off the geometric considerations of the original problem by passing to the Eulerian setting, we cannot do so completely!

Is there an order of convergence that we require for our applications? So far, our only requirements come from (1.5). We will find that we need at least $O(h^2)$ accuracy for the first and second partial derivatives of τ in order to approximate α consistently, although we will not do so by solving (1.5)—more on this in the next section.

Taking all of the foregoing considerations into account, we find that we need an eikonal solver that:

- computes first and second order partials with $O(h^2)$ accuracy,
- can compute T on an unstructured mesh that conforms to $\partial\Omega$,
- uses a compact stencil.

We will find that by using a semi-Lagrangian approach based on discretizing (1.12) will allow us to design a solver that meets these requirements. We develop a solver

with the first and second properties for a general speed of sound in Chapter 3, and a solver with the first two properties in Chapter 4 for $c \equiv 1$.

1.4 Properties of the eikonal equation

In this section, we collect some useful details related to the eikonal equation. The basic fact we assume is that a characteristic $\boldsymbol{\psi}$ of the eikonal equation satisfies (1.12).

The eikonal Euler-Lagrange equations. Let $\boldsymbol{\psi} = \boldsymbol{\psi}(\sigma)$ be a characteristic (or ray) of the eikonal equation, parametrized by arc length. From (1.12), we can define the Lagrangian for the eikonal equation as:

$$\mathcal{L}(\sigma, \boldsymbol{\psi}, \boldsymbol{\psi}') = s(\boldsymbol{\psi}(\sigma))\|\boldsymbol{\psi}'(\sigma)\|. \tag{1.14}$$

The Euler-Lagrange equations are:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\psi}} - \frac{d}{d\sigma}\frac{\partial \mathcal{L}}{\partial \boldsymbol{\psi}'} = 0. \tag{1.15}$$

This turns out to be equivalent to:

$$\|\boldsymbol{\psi}'\|^2 \left(\boldsymbol{I} - \frac{\boldsymbol{\psi}'}{\|\boldsymbol{\psi}'\|} \otimes \frac{\boldsymbol{\psi}'}{\|\boldsymbol{\psi}'\|}\right) \nabla s(\boldsymbol{\psi}) = s(\boldsymbol{\psi}) \left(\boldsymbol{I} - \frac{\boldsymbol{\psi}'}{\|\boldsymbol{\psi}'\|} \otimes \frac{\boldsymbol{\psi}'}{\|\boldsymbol{\psi}'\|}\right) \boldsymbol{\psi}''. \tag{1.16}$$

This follows from vector calculus. If we assume that $\boldsymbol{\psi}$ is parametrized by arc length so that $\|\boldsymbol{\psi}'\| \equiv 1$, it follows that $\boldsymbol{\psi}' \perp \boldsymbol{\psi}''$ along $\boldsymbol{\psi}$, and (1.16) simplifies to:

$$\boldsymbol{\psi}'' = \frac{1}{s}\left(\boldsymbol{I} - \boldsymbol{\psi}' \otimes \boldsymbol{\psi}'\right) \nabla s(\boldsymbol{\psi}). \tag{1.17}$$

So, we can see that a ray bends locally in the direction of ∇s projected into the orthogonal complement of $\boldsymbol{\psi}'$.

The eikonal Hamilton's equations. Computing the Legendre transform of the eikonal equation's Lagrangian (1.14), we find that its Hamiltonian is given by:

$$\mathcal{H}(\boldsymbol{p}, \boldsymbol{x}) = \frac{\|\boldsymbol{p}\|^2 - s(\boldsymbol{x})^2}{2}. \tag{1.18}$$

Equation (1.4) follows from setting $\boldsymbol{p} = \nabla\tau(\boldsymbol{x})$ and requiring $\mathcal{H}(\nabla\tau(\boldsymbol{x}), \boldsymbol{x}) = 0$. Hamilton's equations for the eikonal equation are then given by:

$$\frac{\partial \boldsymbol{x}}{\partial \sigma} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{p}} = \boldsymbol{p}, \qquad \frac{\partial \boldsymbol{p}}{\partial \sigma} = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}} = s(\boldsymbol{x})\nabla s(\boldsymbol{x}). \tag{1.19}$$

A useful consequence of the Hamilton's equation for $\partial\boldsymbol{x}/\partial\sigma$ is that we can conclude that, locally, the rays and gradient of the eikonal point in the same direction. Since $\partial\boldsymbol{x}/\partial\sigma = \boldsymbol{\psi}'$ and $\boldsymbol{p} = \nabla\tau$, we have $\boldsymbol{\psi}' = \nabla\tau(\boldsymbol{\psi})$, hence:

$$\frac{\boldsymbol{\psi}'}{\|\boldsymbol{\psi}'\|} = \frac{\nabla\tau(\boldsymbol{\psi})}{\|\nabla\tau(\boldsymbol{\psi})\|}. \tag{1.20}$$

As an interesting side note, if we take the derivative of (1.20) with respect to σ and simplify, and make use of some of the other relations from this section, we can obtain the eikonal Euler-Lagrange equations given by (1.16).

Taking derivatives along rays. By squaring both sides of eq. (1.4), we have:

$$\nabla\tau(\boldsymbol{x})^\top \nabla\tau(\boldsymbol{x}) = s(\boldsymbol{x})^2. \tag{1.21}$$

This is simple, but note that the left-hand side of (1.21) is a directional derivative. If we assume that $\boldsymbol{\psi}$ has an arc length parametrization, then $\nabla\tau(\boldsymbol{\psi}) = s(\boldsymbol{\psi})\boldsymbol{\psi}'(\sigma)$. Using this relationship and by repeatedly taking the gradient of (1.21) with respect to $\boldsymbol{x}$, we can generate $(d/d\sigma)\nabla^k\tau$ along a ray for all $k \geq 0$.

First, if we take the gradient of (1.21), we get:

$$\nabla^2\tau(\boldsymbol{x})\nabla\tau(\boldsymbol{x}) = s(\boldsymbol{x})\nabla s(\boldsymbol{x}), \tag{1.22}$$

If we let $\boldsymbol{g}(\sigma) = \nabla\tau(\boldsymbol{\psi}(\sigma))$, this gives:

$$\frac{d\boldsymbol{g}}{d\sigma} = \nabla s(\boldsymbol{\psi}(\sigma)). \tag{1.23}$$

Taking the gradient again yields:

$$\nabla^3\tau(\boldsymbol{x})\nabla\tau(\boldsymbol{x}) + \left(\nabla^2\tau(\boldsymbol{x})\right)^2 - s(\boldsymbol{x})\nabla^2 s(\boldsymbol{x}) - \nabla s(\boldsymbol{x}) \otimes \nabla s(\boldsymbol{x}) = 0. \tag{1.24}$$

If we let $\boldsymbol{H}(\sigma) = \nabla^2\tau(\boldsymbol{\psi}(\sigma))$ be the Hessian of τ along $\boldsymbol{\psi}$ (we always denote the Hessian by ∇^2 and the Laplacian by Δ), we have:

$$s\frac{d\boldsymbol{H}}{d\sigma} + \boldsymbol{H}^2 - \frac{1}{2}\nabla^2 s^2 = 0, \tag{1.25}$$

where we write the term involving s a little more compactly as the Hessian of s^2. This is a matrix Riccati equation. We will discuss its solution and how it relates to the solution of (1.5) in Section 1.5.

The fact that we can propagate derivatives of τ along the characteristics is encouraging. As we will see in Chapter 3 and Chapter 4, to get a higher-order solver with a compact stencil, we are predominantly concerned with marching the jet of τ, which is just a collection of values of $\nabla^k\tau$ at $\hat{\boldsymbol{x}}$, for $0 \leq k \leq k_{\max}$. Generating $(d/d\sigma)\nabla^k\tau$ in this way allows us to integrate $\nabla^k\tau$ along $\boldsymbol{\psi}$ after locally minimizing (1.12), which is exactly what we need. In Chapter 3, we develop a method for $k_{\max} = 1$ (i.e., a marcher for τ and $\nabla\tau$); we also march some second derivatives in an indirect way. In Chapter 4 we march τ, $\nabla\tau$, and $\nabla^2\tau$ for $c \equiv 1$.

Curvatures of the wavefront. In chapter 4, we use formulae for scattered amplitudes which depend on the curvature of the wavefront through different sections, and assumes that $c \equiv 1$. The curvature of the wavefront can be easily extracted from $\nabla^2\tau$, which we now derive.

The eikonal gives the first arrival time of a wavefront for each $\boldsymbol{x} \in \Omega$. Hence, for a given arrival time τ_0, the set of points $\boldsymbol{x} \in \Omega$ such that $\tau(\boldsymbol{x}) = \tau_0$ describes the τ_0 wave front as an implicit surface. A variety of formulas for different differential geometric quantites for implicit surfaces can be computed. For the purposes of computing curvatures, we define the unit surface normal of the wavefront to be:

$$\boldsymbol{\nu}(\boldsymbol{x}) = \nabla\tau(\boldsymbol{x})/\|\nabla\tau(\boldsymbol{x})\|. \tag{1.26}$$

We would like to compute the curvature of the wavefront in different plane sections. The following proposition concerning the curvature of an implicit plane curve allows us to do so [57]:

Proposition 1. *Let $f : \mathbb{R}^2 \to \mathbb{R}$ be a scalar field such that the set $\{\boldsymbol{x} \in \mathbb{R}^2 : f(\boldsymbol{x}) = 0\}$ is a plane curve. Then, the curvature at a point $\boldsymbol{x} \in \mathbb{R}^2$ satisfying $f(\boldsymbol{x}) = 0$ is:*

$$\kappa(\boldsymbol{x}) = -\frac{\boldsymbol{t}(\boldsymbol{x})^\top \nabla^2 f(\boldsymbol{x}) \boldsymbol{t}(\boldsymbol{x})}{\|\nabla f(\boldsymbol{x})\|}, \tag{1.27}$$

where $\boldsymbol{t}(\boldsymbol{x})$ is a unit tangent to the curve at $\boldsymbol{x}$.

Now, let $\boldsymbol{q}$ be any unit vector orthogonal to $\boldsymbol{\nu}(\boldsymbol{x})$ (i.e., in the tangent plane of the $\tau(\boldsymbol{x})$ wavefront at $\boldsymbol{x}$). Then, using Proposition 1, we can compute the curvature for the section spanned by $\boldsymbol{\nu}(\boldsymbol{x})$ and $\boldsymbol{q}$ since the wavefront restricted to this section is a plane curve:

$$\kappa_{\boldsymbol{q}}(\boldsymbol{x}) = -\frac{\boldsymbol{q}^\top \nabla^2 \tau(\boldsymbol{x}) \boldsymbol{q}}{s(\boldsymbol{x})}. \tag{1.28}$$

The radius of curvature for this section is just:

$$\rho_{\boldsymbol{q}}(\boldsymbol{x}) = \frac{1}{\kappa_{\boldsymbol{q}}(\boldsymbol{x})}. \tag{1.29}$$

Hence, the center of the osculating sphere contacting the point $\boldsymbol{x}$ is $\boldsymbol{x} + \rho_{\boldsymbol{q}}(\boldsymbol{x})\boldsymbol{\nu}(\boldsymbol{x})$. We note that the sign in (1.27) depends on the orientation of the normal, which is parallel to $\nabla f(\boldsymbol{x})$. We have chosen $\boldsymbol{\nu}(\boldsymbol{x})$ to point in the direction of wavefront propagation. This means that for a point source with $\boldsymbol{x}_{\text{src}} = 0$ and $s \equiv 1$, the surface normal of the spherical wavefront passing through $\boldsymbol{x}$ is $\boldsymbol{\nu}(\boldsymbol{x}) = \boldsymbol{x}/\|\boldsymbol{x}\|$, and for any unit tangent vector $\boldsymbol{q}$ we have $\kappa_{\boldsymbol{q}}(\boldsymbol{x}) = -1/\|\boldsymbol{x}\|$. We also have that $\rho_{\boldsymbol{q}}(\boldsymbol{x}) = -\|\boldsymbol{x}\|$. The osculating sphere touching $\boldsymbol{x}$ is just the spherical wavefront itself, and its center is $\boldsymbol{x} + \rho_{\boldsymbol{q}}(\boldsymbol{x})\boldsymbol{\nu}(\boldsymbol{x}) = 0 = \boldsymbol{x}_{\text{src}}$, as expected.

We will have occasion to compute the principal curvatures of the wavefront. The principal directions are the eigenvectors of the Hessian restricted to the tangent plane of the wavefront at $\boldsymbol{x}$. If we let $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ be the unit vectors coinciding with the two

principal directions, we can use eq. (1.28) to compute κ_1 and κ_2. If $\boldsymbol{\psi}$ is parametrized by arc length, we have:

$$\boldsymbol{\psi}'^{\top}\nabla^2\tau\boldsymbol{\psi}' = D^2_{\boldsymbol{\psi}'}\tau = \frac{d^2\tau}{d\sigma^2} = \frac{ds}{d\sigma}, \tag{1.30}$$

using the fact that $d\tau/d\sigma = s$, which we determined in the section on directional derivatives. Applying (1.28) and (1.30) along $\boldsymbol{\psi}$, and denoting $\boldsymbol{V} = \begin{bmatrix}\boldsymbol{\psi}' & \boldsymbol{v}_1 & \boldsymbol{v}_2\end{bmatrix}$ so that $\boldsymbol{V}^{\top}\boldsymbol{V} = \boldsymbol{I}$, we have the following eigenvalue decomposition of $\nabla^2\tau$:

$$\nabla^2\tau = \boldsymbol{V}\begin{bmatrix}\frac{ds}{d\sigma} & & \\ & -s\kappa_1 & \\ & & -s\kappa_2\end{bmatrix}\boldsymbol{V}^{\top} = \frac{ds}{d\sigma}\boldsymbol{\psi}'\otimes\boldsymbol{\psi}' - s\sum_{i=1}^{2}\kappa_i\boldsymbol{v}_i\otimes\boldsymbol{v}_i. \tag{1.31}$$

Hence, if we happen to know the local ray direction and the principal curvatures of the wavefront at a point, we can reconstruct the Hessian. For a spherical wave with $s \equiv 1$ and $\boldsymbol{x}_{\text{src}} = 0$, this gives:

$$\nabla^2\tau(\boldsymbol{x}) = \frac{1}{\|\boldsymbol{x}\|}\sum_{i=1}^{2}\boldsymbol{v}_i\otimes\boldsymbol{v}_i = \frac{1}{\|\boldsymbol{x}\|}\left(\boldsymbol{I} - \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}\otimes\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}\right), \tag{1.32}$$

as expected.

1.5 Paraxial ray theory

In this section, we give a simplified presentation of paraxial ray theory [98, 97]. The key idea of paraxial ray theory is to develop the equations of motion of a ray tube surrounding a fixed central ray. An auxiliary quantity, the geomeric spreading, is introduced, and it is related to the amplitude along the central ray. See Figure 1.2. The equations of motion for the ray tube are closely related to the evolution of the Hessian of the eikonal in the normal plane of the ray. It is shown that this Hessian satisfies a matrix Riccati equation which is equivalent to the equations of motion of the ray tube. We first present our simplified paraxial ray theory, which evolves the full Hessian along $\boldsymbol{\psi}$, and the standard paraxial ray theory, and show how the two relate.

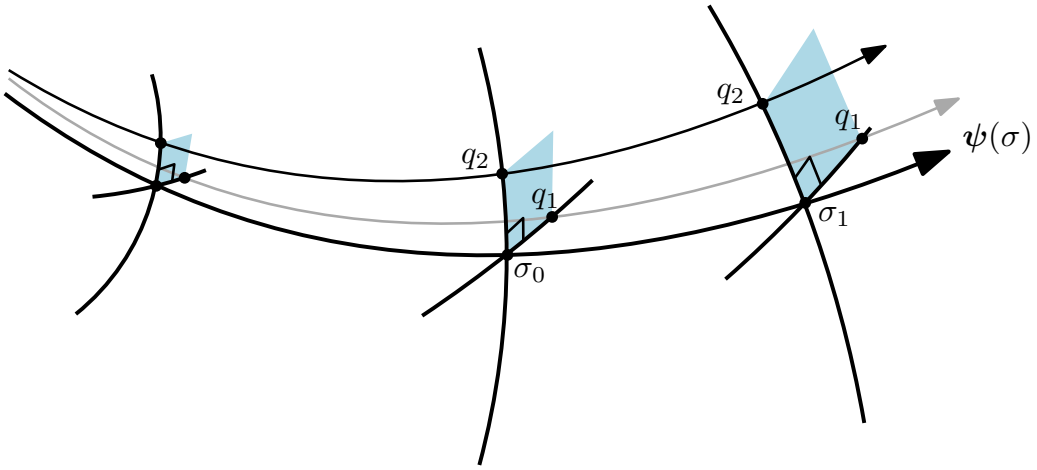


Figure 1.2: A fixed central ray $\boldsymbol{\psi}(\sigma)$, with two nearby rays parametrized using the parameters $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$. In paraxial raytracing, equations of motion for the ray tube are developed, which allow the geometric spreading of the ray tube to be calculated. This provides a means of computing α along $\boldsymbol{\psi}$.

Simplified paraxial ray theory. From Section 1.4, we found that:

$$s\frac{d\boldsymbol{H}}{d\sigma} + \boldsymbol{H}^2 - \frac{1}{2}\nabla^2 s^2 = 0. \tag{1.33}$$

This is a system of ODEs that evolves the Hessian of τ along a fixed ray $\boldsymbol{\psi}$. Assume that we have the initial conditions $\boldsymbol{H}(0) = \nabla^2\tau(\boldsymbol{\psi}(0))$, and would like to compute $\boldsymbol{H}(\sigma)$, for some $\sigma > 0$. The standard way to solve this equation is to introduce the matrices $\boldsymbol{P}(\sigma)$ and $\boldsymbol{Q}(\sigma)$, and make the ansatz $\boldsymbol{H} = \boldsymbol{P}\boldsymbol{Q}^{-1}$. By introducing two matrices, we have an extra degree of freedom. Inserting this ansatz into (1.33) and using the fact that:

$$\frac{d\boldsymbol{Q}^{-1}}{d\sigma} = -\boldsymbol{Q}^{-1}\frac{d\boldsymbol{Q}}{d\sigma}\boldsymbol{Q}^{-1}, \tag{1.34}$$

we get:

$$s\frac{d\boldsymbol{P}}{d\sigma}\boldsymbol{Q}^{-1} - s\boldsymbol{P}\boldsymbol{Q}^{-1}\frac{d\boldsymbol{Q}}{d\sigma}\boldsymbol{Q}^{-1} + \boldsymbol{P}\boldsymbol{Q}^{-1}\boldsymbol{P}\boldsymbol{Q}^{-1} - \frac{1}{2}\nabla^2 s^2 = 0. \tag{1.35}$$

We can easily see that (1.35) holds if the following systems of ODEs for $\boldsymbol{P}$ and $\boldsymbol{Q}$ hold:

$$\frac{d\boldsymbol{Q}}{d\sigma} = \frac{1}{s}\boldsymbol{P}, \qquad \frac{d\boldsymbol{P}}{d\sigma} = \frac{1}{2s}\big(\nabla^2 s^2\big)\boldsymbol{Q} \tag{1.36}$$

hold. So, $\nabla^2\tau$ can be evolved along $\boldsymbol{\psi}$ by solving (1.36).

To compute the amplitude, we consider (1.5) along $\boldsymbol{\psi}$:

$$\alpha(\boldsymbol{\psi}(\sigma))\Delta\tau(\boldsymbol{\psi}(\sigma)) + 2\nabla\tau(\boldsymbol{\psi}(\sigma))^\top \nabla\alpha(\boldsymbol{\psi}(\sigma)) = 0. \tag{1.37}$$

However, noting that $\Delta\tau = \operatorname{tr}(\boldsymbol{H})$, and using $\nabla\tau(\boldsymbol{\psi}) = s(\boldsymbol{\psi})\boldsymbol{\psi}'$ (again, assuming an arc length parametrization of $\boldsymbol{\psi}$), this can be rewritten:

$$\frac{1}{(\alpha\circ\boldsymbol{\psi})(\sigma))}\frac{d(\alpha\circ\boldsymbol{\psi})}{d\sigma} = -\frac{\operatorname{tr}(\boldsymbol{H}(\sigma))}{2s(\boldsymbol{\psi}(\sigma))}. \tag{1.38}$$

If we integrate this over the interval $[0,\sigma]$, we get:

$$\log\alpha(\boldsymbol{\psi}(\sigma)) = \log\alpha(\boldsymbol{\psi}(0)) - \frac{1}{2}\int_0^\sigma \frac{\operatorname{tr}(\boldsymbol{H}(\varsigma))}{s(\boldsymbol{\psi}(\varsigma))}d\varsigma, \tag{1.39}$$

where we assume that α is positive and real.

Next, from (1.31), since the trace of a matrix equals the sum of its eigenvalues, we have:

$$\frac{\operatorname{tr}(\boldsymbol{H}(\sigma))}{s} = \frac{1}{s}\frac{ds}{d\sigma} - \sum_{i=1}^{2}\kappa_i. \tag{1.40}$$

Integrating gives:

$$\int_0^\sigma \frac{\operatorname{tr}(\boldsymbol{H}(\sigma))}{s(\boldsymbol{\psi}(\varsigma))}d\varsigma = \log s(\boldsymbol{\psi}(\sigma)) - \log s(\boldsymbol{\psi}(0)) - \sum_{i=1}^{2}\int_0^\sigma \kappa_i(\boldsymbol{\psi}(\varsigma))d\varsigma. \tag{1.41}$$

Substituting this expression into (1.39) gives and exponentiating both sides gives:

$$\boxed{\frac{\alpha(\boldsymbol{\psi}(\sigma))}{\alpha(\boldsymbol{\psi}(0))} = \sqrt{\frac{s(\boldsymbol{\psi}(0))}{s(\boldsymbol{\psi}(\sigma))}}\exp\left(\frac{1}{2}\sum_{i=1}^{2}\int_0^\sigma \kappa_i(\boldsymbol{\psi}(\varsigma))d\varsigma\right).} \tag{1.42}$$

Let's verify this expression for a point source with $s \equiv 1$. The resulting wavefront is spherical. If $\boldsymbol{\psi}(0) = \boldsymbol{x}_{\text{src}}$ and $\boldsymbol{\psi}$ has an arc length parametrization, then $\kappa_i(\sigma) = -1/\sigma$. Then, for σ_0 and σ_1 such that $0 < \sigma_0 < \sigma_1$ we can compute:

$$\begin{aligned}\frac{\alpha(\boldsymbol{\psi}(\sigma_1))}{\alpha(\boldsymbol{\psi}(\sigma_0))} &= \exp\left(\frac{1}{2}\sum_{i=1}^{2}\int_{\sigma_0}^{\sigma_1}\kappa_i(\boldsymbol{\psi}(\varsigma))d\varsigma\right)\\ &= \exp\left(-\int_{\sigma_0}^{\sigma_1}\frac{d\varsigma}{\varsigma}\right) = \exp(\log\sigma_0 - \log\sigma_1) = \frac{\sigma_0}{\sigma_1}.\end{aligned} \tag{1.43}$$

If we consider this expression for all pairs (σ_0, σ_1), we can see that it recovers $\alpha(\boldsymbol{\psi}(\sigma)) = 1/\sigma$, as expected.

We can also consider (1.42) for a plane wave with $s \equiv 1$. In this case $\kappa_1 \equiv \kappa_2 \equiv 0$, and $\alpha(\boldsymbol{\psi}(\sigma)) \equiv \alpha(\boldsymbol{\psi}(0))$. So, the equations of motion predict that a plane wave has constant amplitude.

Standard paraxial ray theory. We can compare this simplified derivation with the "standard" paraxial ray theory [97]. Following this approach, we let $\boldsymbol{q} = (q_1, q_2)$ be a pair of orthogonal coordinates parametrizing the normal plane of $\boldsymbol{\psi}$ at each point. Then, we define:

$$\tilde{\boldsymbol{H}}(\sigma) = \left.\frac{\partial^2 \tau}{\partial \boldsymbol{q} \partial \boldsymbol{q}^\top}\right|_{\boldsymbol{\psi}(\sigma)}, \tag{1.44}$$

and make the ansatz $\tilde{\boldsymbol{H}}(\sigma) = \tilde{\boldsymbol{P}}(\sigma)\tilde{\boldsymbol{Q}}(\sigma)^{-1}$. The tilde is used to indicate that these quantities are analogous to those defined in the "simplified" paraxial ray theory in the foregoing section, but which live in $\boldsymbol{\psi}$'s normal plane. In the "standard" paraxial ray theory a variety of Taylor approximations are made, resulting in following the equations of motion for $\tilde{\boldsymbol{P}}$ and $\tilde{\boldsymbol{Q}}$:

$$\frac{d\tilde{\boldsymbol{P}}}{d\sigma} = -\frac{1}{c(\boldsymbol{\psi}(\sigma))} \left.\frac{\partial^2 c}{\partial \boldsymbol{q} \partial \boldsymbol{q}^\top}\right|_{\boldsymbol{\psi}(\sigma)} \tilde{\boldsymbol{Q}}(\sigma), \qquad \frac{d\tilde{\boldsymbol{Q}}}{d\sigma} = c(\psi(\sigma))\tilde{\boldsymbol{P}}(\sigma). \tag{1.45}$$

These equations allow us to integrate $\tilde{\boldsymbol{H}}$ along $\boldsymbol{\psi}$. We can then use the results from Section 1.4 to compute $D^2_{\boldsymbol{\psi}'}\tau$ and the cross-derivatives to propagate the complete Hessian. Additionally, (1.5) can be solved along $\boldsymbol{\psi}$ similarly to what was done for the simplified approach, resulting in:

$$\boxed{\frac{\alpha(\boldsymbol{\psi}(\sigma))}{\alpha(\boldsymbol{\psi}(0))} = \sqrt{\frac{s(\boldsymbol{\psi}(\sigma))}{|\det(\tilde{\boldsymbol{Q}}(\sigma))|}}.} \tag{1.46}$$

In the standard paraxial ray theory, we define the geometric spreading to be $J(\sigma) = |\det(\tilde{\boldsymbol{Q}}(\sigma))|$. We will use this version of paraxial ray theory in Chapter 3 and the simplified version in chapter 4.

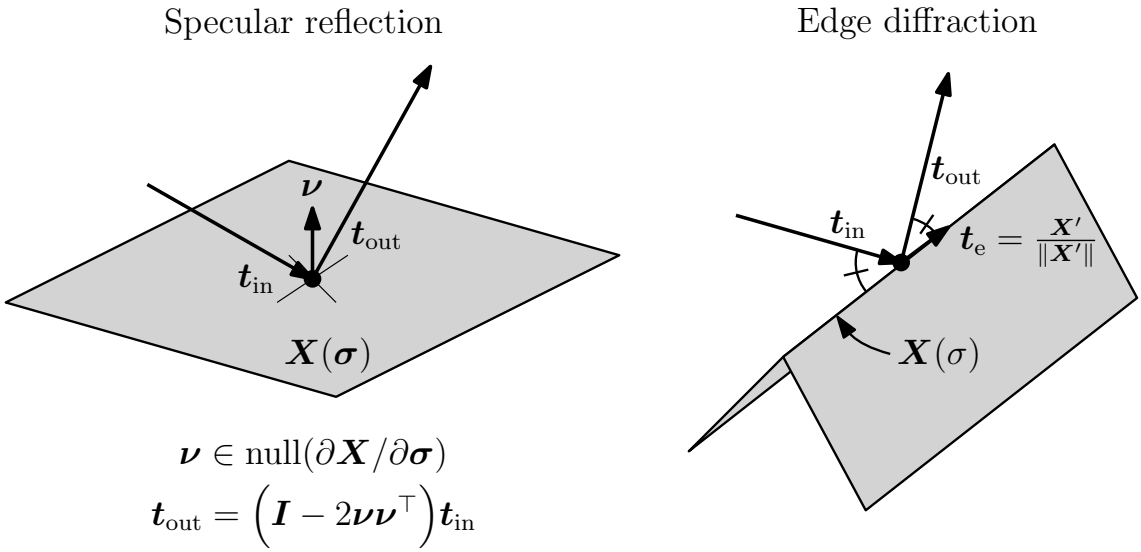


Figure 1.3: The angle conditions for specular reflection and edge diffraction. We denote the reflecting surface's and diffracting edge's parametrizations by $\boldsymbol{X}$, the tangent vector of the incident and emitted ray by $\boldsymbol{t}_{\text{in}}$ and $\boldsymbol{t}_{\text{out}}$, respectively, the surface normal of the reflecting surface by $\boldsymbol{\nu}$, and the edge's tangent vector by $\boldsymbol{t}_e$. Computing the eikonal of a scattered field is straightforward: restrict τ_{in} to the scattering feature and solve a new eikonal equation with those BCs to compute τ_{out}.

1.6 Multiple arrivals and the geometric theory of diffraction

Keller's geometric theory of diffraction (GTD) augments geometric optics/acoustics with an asymptotic approximation of diffraction phenomena [69]. Later, Kouyoumjian and Pathak derived a version of GTD—called the uniformed theory of diffraction (UTD)—using matched asymptotics (boundary layer theory) to extend the validity GTD's nonuniform approximation [75]. We will refer to GTD and UTD together as GTD, since Keller's original development is more fundamental.

The generalized Fermat principle and BCs for the eikonal. The starting point for GTD is replacing Fermat's principle with a generalized Fermat's principle with constraints inferred from the environment. The goal is to find an extremal of (1.12) where some point along ψ is constrained to lie in a subset $S \subseteq \partial\Omega$ with constant

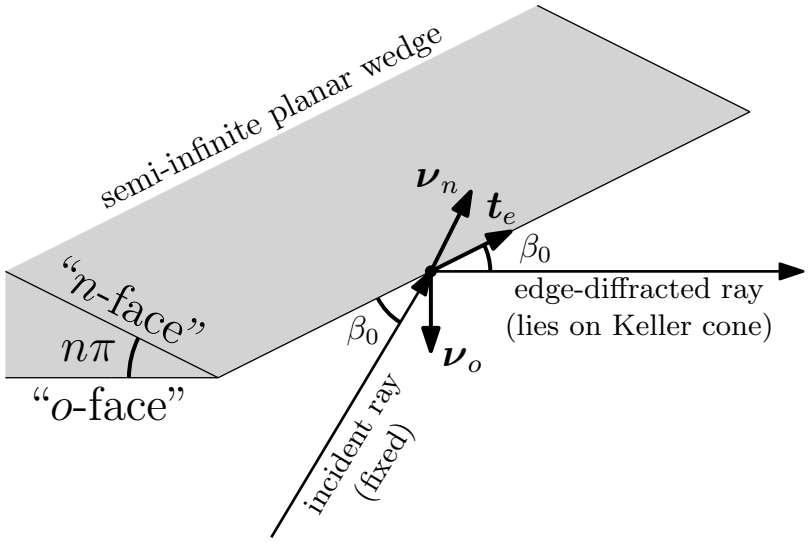


Figure 1.4: Edge-diffraction from a semi-infinite planar wedge. The diffraction coefficient and the diffracted amplitude depend on the local geometry of the wedge at the point of diffraction. A ray which strikes an edge splits into a cone of diffracted rays, each of which makes the same angle with the edge at the point of diffraction. This cone is referred to as Keller's cone.

codimension. That is, the point could be required to lie on a smooth edge, facet, or vertex.

Since we think of τ as a travel time, and if we let τ_{in} and τ_{out} denote the incident and scattered fields, respectively, then continuity gives the simple condition:

$$\tau_{\text{in}}(\boldsymbol{x}) = \tau_{\text{out}}(\boldsymbol{x}), \qquad \boldsymbol{x} \in S. \tag{1.47}$$

Let $k = \dim(S)$, let E be an open subset of $\mathbb{R}^k$, let $F \subseteq S$ be a relatively open subset of S, let $\boldsymbol{x}_0 \in S$, and let $\boldsymbol{X} : E \to F$ be a parametrization of S valid about $\boldsymbol{x}_0 = \boldsymbol{X}(0)$. In $\mathbb{R}^3$, if $k = 2$, we have a reflecting surface; if $k = 1$, we have a diffracting edge; and if $k = 0$, we have a diffracting corner/vertex. Then, with $\boldsymbol{x} = \boldsymbol{X}(\boldsymbol{\sigma})$, taking the gradient of (1.47) and evaluating at $\boldsymbol{\sigma} = 0$ gives:

$$\left.\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{\sigma}}\right|_{\boldsymbol{\sigma}=0}^{\top} \frac{\partial \tau_{\text{in}}}{\partial \boldsymbol{x}} = \left.\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{\sigma}}\right|_{\boldsymbol{\sigma}=0}^{\top} \frac{\partial \tau_{\text{out}}}{\partial \boldsymbol{x}}. \tag{1.48}$$

In $\mathbb{R}^3$, we have two cases where this applies. If $k = 1$, then $\partial \boldsymbol{X}/\partial \boldsymbol{\sigma}$ is a tangent vector of S at $\boldsymbol{X}(\boldsymbol{\sigma})$, and (1.48) stipulates that the angle between $\boldsymbol{\psi}'_{\text{in}}$ and the tangent

space at $\boldsymbol{x}_0$ and the angle between $\boldsymbol{\psi}'_{\text{out}}$ and the tangent space at $\boldsymbol{x}_0$ is the same. This is the well-known geometric condition defining Keller's cone of rays (see Figure 1.4). It implies that a ray incident on a singular edge of a surface in 3D generates a bundle of rays with initial ray directions lying on the surface of a cone symmetric about the tangent space at $\boldsymbol{x}_0$ (Keller's cone). We note that while (1.48) provides on a single condition for fixing $\nabla\tau_{\text{out}}$, we have another provided by (1.4) itself; the remaining degree of freedom is reflected by the fact that Keller's cone of rays can be parametrized by a single variable.

Next, if $k = 2$, then $\partial\boldsymbol{X}/\partial\boldsymbol{\sigma}$ spans the tangent plane of a smooth two-dimensional subset of $\partial\Omega$. Equation (1.48) and (1.4) provide a full set of constraints to uniquely determine $\nabla\tau_{\text{out}}$. Since the generalized Fermat's principle is minimized over paths in Ω, we can then see that (1.48) generates the usual specular reflection condition. That is, $\nabla\tau_{\text{out}}$ is just $\nabla\tau_{\text{in}}$ reflected over the tangent plane of S at $\boldsymbol{x}_0$.

Lastly, we could also consider the case $k = 0$. In this case, $\boldsymbol{x}_0$ is constrained to lie on a vertex, which does not have a nonempty relative interior. The intuition here is straightforward: any path will be a stationary path, since there is no way to perturb the point $\boldsymbol{x}_0$ without violating the constraint. This means that the vertex will act like a point source. It turns out that in this case, there will be a corresponding $O(\omega^{-1})$ reduction in amplitude, which is of the same order as the error in the GA approximation. For this reason, vertex diffraction is typically ignored, and is usually viewed as an unnecessary complication.

Boundary conditions for the reflected amplitude. The BCs connecting α_{in} and α_{out} are inferred from the BCs for the Helmholtz equation for P. Sound hard scattering is determined by the Neumann BC:

$$\left.\frac{\partial P}{\partial \boldsymbol{\nu}}\right|_S \equiv 0, \tag{1.49}$$

where $\boldsymbol{\nu}$ is the surface normal of S. The Dirichlet BCs:

$$P|_S \equiv 0, \tag{1.50}$$

correspond to scattering from a sound soft body. General impedence BCs are given by the Robin BCs:

$$\left(\frac{\partial P}{\partial \boldsymbol{\nu}} + ikzP\right)\Big|_S \equiv 0. \tag{1.51}$$

In this work, we will focus on sound hard BCs. The other two BCs do not present any additional technical difficulties, but can be used in a later work to model different phenomena more accurately.

In the rest of this section, we will summarize the GA BCs for the amplitude for reflection and diffraction. We will assume that $c \equiv 1$. Note that the preceding section on paraxial ray theory was developed to support a variable speed of sound. The results summarized in this section (including basic UTD results) are focused on the case $c \equiv 1$. Even in this case, the results are fairly complicated, and applying them in the semi-Lagrangian setting requires some work. While part of Chapter 3 concerns applying paraxial ray theory to compute the amplitude for a point source in free space, the focus of chapter 4 is using the material from this section to compute the amplitude in an environment with complicated obstacles. Developing (or tracking down) versions of these results that are compatible with varying c and combining the results of chapters 3 and 4 is an ambitious task which we leave for future work.

Again, assuming that $c \equiv 1$, and letting $\boldsymbol{x}_0$ be the point of reflection, and $\boldsymbol{x}$ be a target point away from the reflector, the reflected amplitude satisfies:

$$\boxed{\alpha_{\text{out}} = R\sqrt{\frac{\rho_1\rho_2}{(\rho_1+\sigma)(\rho_2+\sigma)}}\alpha_{\text{in}}}. \tag{1.52}$$

Here, ρ_1 and ρ_2 are the principal curvatures of the reflected wavefront at the point of reflection, and σ is the arc length parameter connecting $\boldsymbol{x}_0$ and $\boldsymbol{x}$ by $\boldsymbol{\psi}_{\text{out}}$. The factor R is the sound-hard reflection cofficient, which is predicted to be 1 in our ideal setup.

In practice, R is used to model loss due to different types of materials. E.g., R will be small for a material like gypsum board which causes more acoustic damping, and high for a material like concrete which transmits relatively more acoustic energy upon reflection. Tables of reflection coefficients for different frequencies are available in the literature and are determined empirically. As a reminder: all of what is being stated holds on a per-frequency basis, since we are considering a particular time-harmonic solution of (1.2).

We also note the obvious connection with the geometric spreading J. Indeed, if c is constant, then:

$$\frac{J(\sigma)}{J(\sigma_0)} = \frac{(\rho_1 + \sigma - \sigma_0)(\rho_2 + \sigma - \sigma_0)}{\rho_1 \rho_2}. \tag{1.53}$$

Comparing this with the results in Section 1.5, we have:

$$\frac{\alpha_{\text{out}}(\boldsymbol{x})}{\alpha_{\text{in}}(\boldsymbol{x}_0)} = \sqrt{\frac{J_{\text{in}}(\sigma_0)}{J_{\text{out}}(\sigma)}} = \sqrt{\frac{\rho_1 \rho_2}{(\rho_1 + \sigma - \sigma_0)(\rho_2 + \sigma - \sigma_0)}}. \tag{1.54}$$

Modulo the reflection coefficient R (which, again, should equal unity in an ideal setting), this matches (1.52) exactly. This gives us some assurance that, although we focus on $c \equiv 1$ in Chapter 4, the skeleton of our approach can be extended to handle a varying speed of sound.

The diffracted amplitude. Determining the BCs for the diffracted amplitude is a vastly more complex undertaking than what was sketched in the last section. In this short section, we summarize well-known UTD results available in more detail elsewhere [75, 89, 85].

In this work, we focus exclusively on diffraction from a sound-hard wedge with linear facets. The diffracted amplitude satisfies:

$$\boxed{\alpha_{\text{out}}(\boldsymbol{x}) = D\sqrt{\frac{\rho_e}{\sigma(\rho_e + \sigma)}}\alpha_{\text{in}}(\boldsymbol{x}_0),} \tag{1.55}$$

where $\boldsymbol{x}$ is again the observation point, $\boldsymbol{x}_0$ is the point of diffraction, and σ is the arc length of the segment of $\boldsymbol{\psi}_{\text{out}}$ connecting them (i.e., $\sigma = \|\boldsymbol{x} - \boldsymbol{x}_0\|$. In this case, ρ_e is

the radius of curvature of the diffracted wavefront in the plane of diffraction (that is, in the plane spanned by the tangent vector of the diffracting edge and $\boldsymbol{\psi}'$).

We will start by laying out the general formula for diffraction from a sound-hard curved wedge to give some context, before simplifying to diffraction from a sound-hard wedge with linear facets. In what follows, a lot of specialized notation will be introduced. As much as possible we try to match the usage in the literature. We let n be such that the exterior angle of the wedge is given by $(2-n)\pi$. We let $\boldsymbol{t}_e$ denote the tangent vector of the edge at $\boldsymbol{x}_0$ and let β_0 be the angle between $\boldsymbol{\psi}'_{\text{in,out}}$ and $\boldsymbol{t}_e$. In the UTD literature, the two faces of the wedge are arbitrarily labeled the o-face and the n-face. Then, we let φ_{in} be the angle that $\boldsymbol{\psi}'_{\text{in}}$ makes with the o-face, measured in the edge's normal plane at $\boldsymbol{x}_0$; likewise, we let φ_{out} denote the corresponding angle for $\boldsymbol{\psi}'_{\text{out}}$. Note that $\varphi_{\text{in}}, \varphi_{\text{out}} \in [0, n\pi]$.

To define D, we need to introduce several auxiliary functions. We let:

$$\beta^{\pm} = \varphi_{\text{out}} \pm \varphi_{\text{in}} \in [-n\pi, n\pi], \tag{1.56}$$

and:

$$N^{\pm} = \arg\min_{N^{\pm}\in\mathbb{Z}} \left|\beta^{\pm} - 2\pi n N^{\pm} \pm \pi\right| \in \{-1, 0, 1\}. \tag{1.57}$$

Then, we define the auxiliary function:

$$a^{\pm}(\beta) = 2\cos^2\left(\frac{2n\pi N^{\pm} - \beta}{2}\right), \tag{1.58}$$

and the transition function:

$$F(x) = 2i\sqrt{x}\int_{\sqrt{x}}^{\infty} e^{-it^2}dt. \tag{1.59}$$

Additionally, we define three distance parameters:

$$\begin{aligned}
L^{\text{in}} &= \frac{\sigma(\rho_e^{\text{in}}+\sigma)\rho_1^{\text{in}}\rho_2^{\text{in}}}{\rho_e^{\text{in}}(\rho_1^{\text{in}}+\sigma)(\rho_2^{\text{in}}+\sigma)}\sin^2\beta_0,\\
L^{\text{refl},o} &= \frac{\sigma(\rho_e^{\text{refl},o}+\sigma)\rho_1^{\text{refl},o}\rho_2^{\text{refl},o}}{\rho_e^{\text{refl},o}(\rho_1^{\text{refl},o}+\sigma)(\rho_2^{\text{refl},o}+\sigma)}\sin^2\beta_0,\\
L^{\text{refl},n} &= \frac{\sigma(\rho_e^{\text{refl},n}+\sigma)\rho_1^{\text{refl},n}\rho_2^{\text{refl},n}}{\rho_e^{\text{refl},n}(\rho_1^{\text{refl},n}+\sigma)(\rho_2^{\text{refl},n}+\sigma)}\sin^2\beta_0,
\end{aligned} \tag{1.60}$$

where ρ_e^{in} is the radius of curvature of the incident wavefront in the plane of incidence (the plane spanned by $\boldsymbol{t}_e$ and $\boldsymbol{\psi}'_{\text{in}}$), ρ_1^{in} and ρ_2^{in} are the principal radii of curvature of the incident wave at the point of diffraction, and β_0 is as before. The remaining parameters are more delicate. Recall that we are considering a curved wedge for the moment: both facets and the set S corresponding to the diffracting edge itself may have nonzero curvature. Let $\boldsymbol{\nu}_e$ be the normal for the edge at the point of diffraction, determined from the edge's osculating circle, and let $\boldsymbol{\nu}_o$ and $\boldsymbol{\nu}_n$ be the surface normal of the o-face and n-face, respectively, and let a_e be the radius of curvature of the edge itself, both at $\boldsymbol{x}_0$. First, denoting o and n by $*$, the radius of curvature $\rho_e^{\text{refl},*}$ is the radius of curvature of the $*$-reflected wavefront in the plane of reflection, which is given by:

$$\frac{1}{\rho_e^{\text{refl},*}} = \frac{1}{\rho_e^{\text{in}}} - \frac{2\boldsymbol{\nu}_e^\top \boldsymbol{\nu}_* \boldsymbol{\nu}_*^\top \boldsymbol{\psi}'_{\text{in}}}{|a_e| \sin^2 \beta_0}. \tag{1.61}$$

Along the same lines, $\rho_1^{\text{refl},*}$ and $\rho_2^{\text{refl},*}$ are the principal radii of curvature of the $*$-reflected waves at the point of diffraction.

With these parameters defined, we can define the diffraction coefficient as follows:

$$\boxed{D(L^{\text{in}}, L^{\text{refl},o}, L^{\text{refl},n}, \varphi_{\text{in}}, \varphi_{\text{out}}, \beta_0, n) = D_1 + D_2 + R \cdot (D_3 + D_4),} \tag{1.62}$$

where R is the empirically determined sound-hard reflection coefficient, and:

$$\begin{aligned}
D_1 &= \frac{-e^{-i\pi/4}}{2n\sqrt{2\pi k}\sin\beta_0} \cot\left(\frac{\pi+\beta^-}{2n}\right) F\big(kL^{\text{in}} a^+(\beta^-)\big), \\
D_2 &= \frac{-e^{-i\pi/4}}{2n\sqrt{2\pi k}\sin\beta_0} \cot\left(\frac{\pi-\beta^-}{2n}\right) F\big(kL^{\text{in}} a^-(\beta^-)\big), \\
D_3 &= \frac{-e^{-i\pi/4}}{2n\sqrt{2\pi k}\sin\beta_0} \cot\left(\frac{\pi+\beta^+}{2n}\right) F\big(kL^{\text{refl},n} a^+(\beta^+)\big), \\
D_4 &= \frac{-e^{-i\pi/4}}{2n\sqrt{2\pi k}\sin\beta_0} \cot\left(\frac{\pi-\beta^+}{2n}\right) F\big(kL^{\text{refl},o} a^-(\beta^+)\big).
\end{aligned} \tag{1.63}$$

The purpose of D is to patch together the incident and reflected fields continuously using the diffracted field.

Equation (1.62) gives a very general formula which allows for curved wedges and an incident wavefront with arbitrary principal curvatures. Similar to (1.61), $\rho_1^{\text{refl},*}$

(resp., $\rho_2^{\text{refl},*}$) can be written as ρ_1^{in} (resp., ρ_2^{in}) with a correction involving a term depending on the curvature of the wedge subtracted, where the correction is zero if the faces of the wedge are flat. Then, for a flat wedge, $L^{\text{in}} = L^{\text{refl},o} = L^{\text{refl},n}$, and (1.62) simplifies to:

$$\boxed{D(L^{\text{in}}, \varphi_{\text{in}}, \varphi_{\text{out}}, \beta_0, n) = D_1 + D_2 + R \cdot (D_3 + D_4),} \tag{1.64}$$

where the expressions for D_1, D_2, D_3, and D_4 given by (1.63) are correspondingly simplified.

Multiple-arrival BCs in the semi-Lagrangian setting. The formula for D must be evaluated throughout Ω, but depends on quantities which must be evaluated at the point of diffraction. (In fact, the same is true of the expression for the reflected amplitude.) If we were doing simple raytracing, this would present no issue. Let $\boldsymbol{x} \in \Omega$ be a point such that $\boldsymbol{\psi}_{\text{out}}(\sigma) = \boldsymbol{x}$, and let $\boldsymbol{x}_0 = \boldsymbol{\psi}_{\text{out}}(0)$ be the point of reflection or diffraction, as before, and let f be a function defined on S, such as ρ_e^{in}, or one of the other radii of curvature. Then, we will need to be able to evaluate:

$$f_0(\boldsymbol{x}) = f(\boldsymbol{x}_0(\boldsymbol{x})). \tag{1.65}$$

Our approach throughout this dissertation is to focus on semi-Lagrangian marching methods for solving the eikonal equation. This approach allows us to transport various quantities along the solution. The algorithm for doing so is simple and runs in $O(N)$ time, which is optimal. This will be made more explicit in Chapter 4. It will turn out that explicitly evaluating the map $\boldsymbol{x}_0 = \boldsymbol{x}_0(\boldsymbol{x})$ is complicated and difficult, but evaluating $f_0 = f \circ \boldsymbol{x}_0$ directly is much more manageable.

1.7 Related work

Beam and frustum tracing, other GA algorithms

Modeling room acoustics by solving a tree of eikonal problems bears some similarity to beam tracing [64], which is a purely geometric approach that traces convex beams from a point source to the distinct reflectors and edge diffractors visible in the scene. Beam tracing has been augmented with edge diffraction and applied to room acoustics [54]. Beam tracing is itself an improvement over the image source method which uses virtual sources to model reflection across polygonal surfaces [5, 21]. More recent methods try to do an approximate form of beam tracing to handle the proliferation of beams in a complicated scene [33, 84]. Each of these are GA methods, and, critically, rely on the assumption that the speed of sound is constant to apply geometric algorithms. There are many other approaches to acoustic raytracing using GA [112].

Our approach differs as follows:

1. We develop eikonal solvers for a spatially varying speed of sound in Chapter 2 and Chapter 3. Although we focus on $c \equiv \text{const}$ in Chapter 4, there is a clear path to incorporating a varying speed of sound in the future. By allowing c to vary spatially, we can easily trace curved beams.

2. Similarly, our approach paves the way to handling caustics. Caustics occur exactly where the geometric spreading of a ray tube vanishes (see Section 1.5). Using the geometric spreading, caustics can be detected and handled by introducing secondary eikonal problems describing the multiply-arriving rays formed by the caustics. Note that this can happen when a ray field reflects or diffracts from a curved interface. To the best of our knowledge, existing GA methods do not typically incorporate caustics.

Precomputed sound propagation, parametric encoding

Offline sound propagation is often done in the time-domain. Finite-difference time-domain (FDTD) methods, in particular those based on two-step schemes, are popular [63, 18]. There are well-known finite volume methods that can be used for linear acoustics [79], and they have received some attention for room acoustics simulations [19, 20]. High-order pseudospectral solvers for the wave equation have been developed for sound propagation [108, 111]. There are also methods using the boundary element method which can be accelerated using the fast multipole method [74, 61]. Still other approaches treat physical sound synthesis as, effectively, a sound propagation problem, and solve the wave equation in order to simulate complex and realistic sound effects for animation [138].

The complexity of direct simulation of wave physics in the time or frequency domain depends strongly on the highest frequency simulated, which is a consequence of the sampling theorem. In the finite element (FEM) literature for solving the Helmholtz equation, this is referred to as the pollution effect [10]. As discussed in Section 1.1, GA removes this limitation by factoring the oscillatory solution of the Helmholtz equation into two slowly varying functions; the trade-off is that a separate set of PDEs must be solved for each field of rays.

Time-domain methods can be used to synthesize RIRs by solving the wave equation for a point source, or—using Helmholtz reciprocity—a "point listener". Either way, the solution encodes a field of point-to-point RIRs for a fixed source/listener location. Computing the full set of pairwise RIRs in the time-domain is time and memory intensive. Recent approaches instead parametrically encode the RIR using a small number of important psychoacoustic parameters [109, 110, 32, 31]. An interesting feature of this approach is that these psychoacoustic parameters are predominantly GA quantities. We are particularly interested in how the techniques developed in Chapter 4 can be used to directly compute the field of parametrically encoded RIRs,

as well as how to compute RIRs for point listeners.

Eulerian geometric optics

Repeatedly solving the eikonal equation to compute multiple arrivals is not a new idea, and has been explored mainly in computational geophysics, where it goes under the name Eulerian geometric optics [15, 46] (EGO). Some techniques developed for EGO have been used for computer graphics [66]. EGO was mainly focused on using GTD to resolve the complicated behavior of caustics and shadow zones that occur with wave propagation in stratified media [12, 13]. Research stopped because it did not compare favorably with raytracing. Although the overall structure of our approach is similar to EGO, there are many differences, between both the individual details of the algorithms, and the nature of the problem being solved. The speed of sound varies slowly in computational room acoustics, with the main challenge being geometric in nature. Our view is that this is a better fit for EGO. We also emphasize that the research on EGO is fragmented and incomplete, and that the developments presented here have not been developed already in the EGO literature, and are largely complementary to them. "Backporting" them to computational geophysics may be of interest.

Chapter 2

Ordered line integral methods and multiple arrivals in 2D

This Chapter is based on the paper "Ordered line integral methods for solving the eikonal equation" [101]. We view this work as an important study that laid the way for the development of the higher-order eikonal solvers presented in Chapter 3 and Chapter 4.

2.1 Introduction

We develop fast, memory efficient, and accurate solvers for the eikonal equation, a nonlinear hyperbolic PDE encountered in high-frequency wave propagation [46] and the modeling of a wide variety of problems in computational and applied science [118], such as photorealistic rendering [66], constructing signed distance functions in the level set method [95], solving the shape from shading problem [73, 103, 45], travel-time computations in numerical modeling of seismic wave propagation [118, 99, 71, 135, 137], and others. We are motivated primarily by problems in high-frequency acoustics [105], which are key to enabling a higher degree of verisimilitude in virtual reality simulations (see [109, 110] for a cutting-edge time-domain approach which is

useful up to moderate frequencies). Current approaches to acoustics simulations rely on methods whose complexity depends on the highest frequency of the sound being simulated. For moderately high-frequency wave propagation problems, the eikonal equation comes about as the first term in an asymptotic WKB expansion of the Helmholtz equation and corresponds to the first arrival time of rays propagating under geometric optics, although approaches for computing multiple arrivals exist [53].

In this work, we develop direct solvers for the eikonal equation which are fast and accurate, particularly in 3D. We develop a family of algorithms which approach the problem of efficiently computing updates in 3D in different ways. This family of algorithms is analyzed and extensive numerical studies are carried out. Our algorithms are semi-Lagrangian, using information about local characteristics to reduce the work necessary to get an accurate result. They are competitive with existing direct solvers for the eikonal equation and generalize to higher dimensions and related equations, such as the static Hamilton-Jacobi equation. In fact, this research was done in tandem with research on the ordered line integral methods for the quasipotential of nongradient stochastic differential equations (SDEs) [41, 40, 140]. Due to the relative simplicity of the eikonal equation, the algorithms presented here are more amenable to analysis, allowing us to obtain theoretical results that justify our experimental findings.

Results

Different numerical methods have been proposed for the solution of the eikonal equation; generally, there are direct solvers and iterative solvers. The most popular direct solvers are based on Dijkstra's algorithm ("Dijkstra-like" solvers) [134, 118], and the most popular iterative method is the fast sweeping method [132, 143]. In this work, we develop a family of Dijkstra-like solvers for the eikonal equation in 2D and 3D, similar to the fast marching method (FMM) or ordered upwind methods

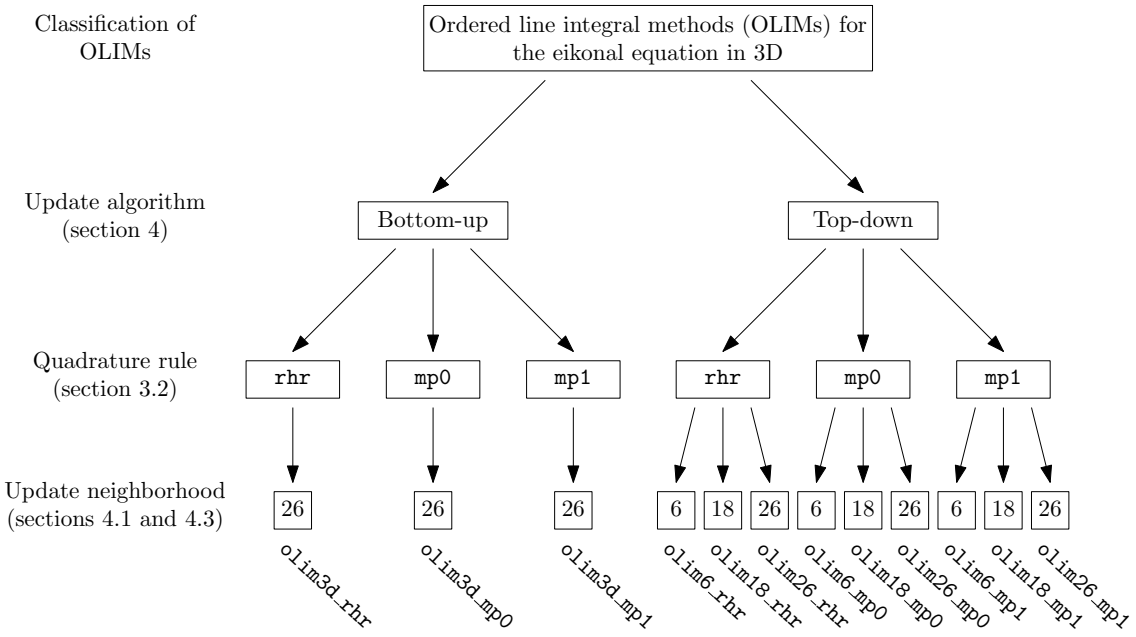


Figure 2.1: *The family of Dijkstra-like solvers designed and studied in this work.* We refer to these as ordered line integral methods (OLIMs). There are three ways of parametrizing the family: by selecting an update algorithm, by selecting a quadrature rule, and by (in the case of the *top-down* update algorithm, by selecting a neighborhood size. Sections in the text that explain these choices in detail are indicated. A shorthand notation for referring to each parametrized algorithms is listed for each algorithm that is involved in numerical tests (e.g., `olim3d_mp0`).

(OUMs) [118, 120]. In constrast to the FMM and OUMs that use finite difference schemes, our solvers come about by discretizing and minimizing the action functional for the eikonal equation. The proposed family of algorithms is parameterized by a choice of update algorithm (*bottom-up* or *top-down*), quadrature rule (a righthand rule (`rhr`), a simplified midpoint rule (`mp0`), and a midpoint rule (`mp1`)), and, in the case of *bottom-up*, a neighborhood size (6, 18, or 26 points in 3D). See figure 2.1.

Our *bottom-up* and *top-down* update algorithms represent two separate approaches to minimizing the number of triangle and tetrahedron updates that need to be done in 3D, while the quadrature rules represent a trade-off between speed and accuracy of the solver. The simplified midpoint rule (`mp0`) is a sweet spot that requires extra theoretical justification, which we provide. Overall, our goal is to explore the relevant

algorithm design trade-offs in 3D and find which solver performs best. Our conclusion is that, in 3D, `olim3d_mp0` is the best overall, and our results are oriented towards supporting this claim.

Additionally, we modify our algorithms to solve the additively factored eikonal equation [81]: to enhance accuracy, we solve the locally factored eikonal equation near point sources, which recovers the global $O(h)$ error convergence expected from a first-order method, where $h > 0$ is the uniform spacing between grid points. This fixes the degraded $O(h \log h^{-1})$ convergence often associated with point source eikonal problems [106] (see [143] for a proof of this error bound).

Our main results follow:

- **For 3D problems, we develop two separate update algorithms:** a *bottom-up* (`olim3d`) algorithm, and a *top-down* algorithm (`olim`K, where $K = 6, 18, 26$ is the size of neighborhood used). Each algorithm locally updates a grid point by performing a minimal number of triangle or tetrahedron updates. Depending on the quadrature rule, each update is calculated by solving a system of nonlinear equations either directly (`rhr` and `mp0`) or iteratively (`mp1`).

- **We prove theorems relating our quadrature rules, rigorously justifying the `mp0` rule.** These results support our case that it is superior to the `mp1` rule. We note that this work was done in tandem with research on ordered line integral methods for computing the quasipotential 3D for nongradient SDEs [41, 140, 40]. Unlike the quasipotential, the eikonal equation is simple enough to allow us to analyze and justify our algorithms. We are also able to obtain simpler solution methods and establish performance guarantees.

- **We conduct numerical experiments on test problems with analytic solutions.** The test problems include point source problems for different slowness (index of refraction) functions, and multiple point source problems with a

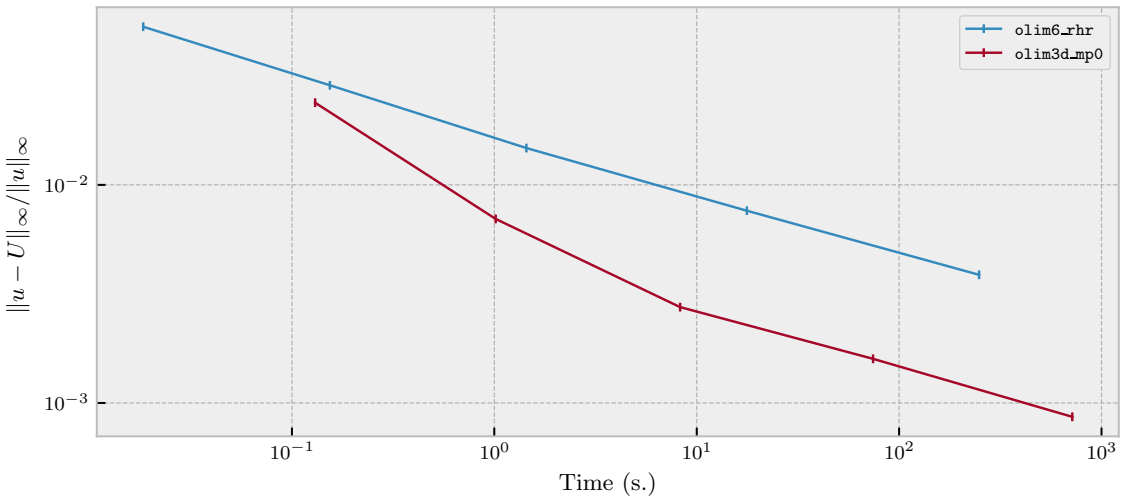


Figure 2.2: *Comparing the relative ℓ_∞ errors of* ***olim3d_mp0*** *and* ***olim6_rhr****.* For the multiple point source problem in section 2.12 with the domain $\Omega = [0,1]^3$ discretized in each direction into $N = 2^p + 1$ (where $p = 5, \ldots, 9$), the total number of grid points is N^3.

linear speed function. All of these have analytical solutions, which we use as a ground truth. We also test our

- **We show that a significant improvement in accuracy is gained over the equivalent of the standard fast marching method in 3D,** `olim6_rhr`**.** Only a modest slowdown is incurred using our general framework, indicating that our approach is competitive. See figure 2.2 to see the improvement of `olim3d_mp0` over `olim6_rhr`, and see section 4.12 for more details.

- **We use Valgrind [92] to profile our implementation.** Our results indicate that the time spent sorting the heap used to order nodes on the front is negligible for all practical problem sizes. Since our solvers otherwise run in $O(N^n)$ time, where n is the dimension of the domain, we suggest that the $O(N^n \log N)$ cost of the algorithm is pessimistic. Memory access patterns play a much more significant role in scaling.

2.2 Ordered line integral methods for the eikonal equation

To numerically solve eq. 1.4, first let $\mathcal{G} = \{p_i\} \subseteq \Omega$ be the set or grid of nodes where we would like to approximate the true solution u with a numerical solution $U : \mathcal{G} \to \mathbb{R}_+$. Additionally, for each node $p \in \mathcal{G}$, define a set of neighbors, $\texttt{nb}(p) \subseteq \mathcal{G} \setminus \{p\}$. Typically—for the FMM, for instance—$\mathcal{G}$ is taken to be a subset of a lattice in $\mathbb{R}^n$ and $\texttt{nb}(p)$ to be each node's $2n$ nearest neighbors. We also define the set of boundary nodes, $\texttt{bd} \subseteq \mathcal{G}$. It may happen that the set $\texttt{bd}$ and D do not coincide (e.g., D could be a curve which does not intersect any points in $\mathcal{G}$); to reconcile this difference, the initial value of $U(p)$ for each $p \in \texttt{bd}$ must take $g = u|_D$ into account in the best way possible. This problem has been approached in different ways, and is not the focus of the present work [35].

Throughout, we make several simplifying assumptions.

- All boundary nodes coincide with grid points: $\texttt{bd} = D \subseteq \mathcal{G}$.
- The grid $\mathcal{G}$ is a regular, uniform grid (a subset of a regular, uniform square lattice in 2D or cubic lattice in 3D). We denote grid nodes by $x \in \mathcal{G}$.
- When numerically computing a new value at a grid point $\hat{x} \in \mathcal{G}$, we transform the neighborhood to the origin and scale the vertices so that they have integer values. The transformed update node is labeled $\hat{p}$. See section 2.3 for a detailed explanation.

To write down a generic Dijkstra-like algorithm, there are several pieces of information which need to be kept track of. A data structure called `front` tracks `trial` nodes while the solver runs (typically an array-based heap). For each node p, apart from the current value of $U(p)$, the most salient piece of information is its state, writ-

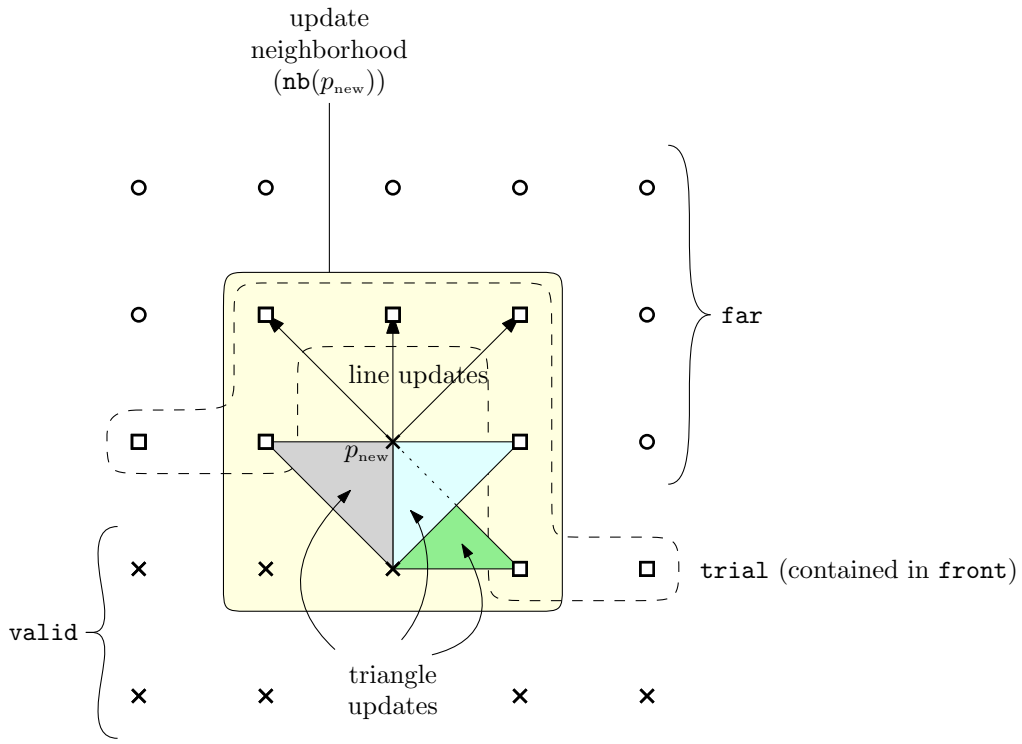


Figure 2.3: *An overview of a Dijkstra-like algorithm for solving the eikonal equation (eq. 1.4) in 2D.* See alg. 1 for details. Nodes are labeled by state so that $\circ$ = `far`, $\square$ = `trial`, and $\times$ = `valid`. In this diagram, the node p_{new} has been removed from `front` and had its state set to `valid`. All `far` nodes in $\texttt{nb}(p_{\text{new}})$ are set to `trial`, and then all `trial` nodes in $\texttt{nb}(p_{\text{new}})$ are updated. The updates are depicted: there are three line updates and three triangles, since it is only necessary to perform updates that involve p_{new}. The OLIM shown here is `olim8`. In 3D, there would also be tetrahedron updates.

ten $p.\texttt{state} \in \{\texttt{valid}, \texttt{trial}, \texttt{far}\}$. To fix ideas, consider the following high-level Dijkstra-like algorithm:

Algorithm 1 A generic Dijkstra-like algorithm for solving the eikonal equation.

1. For each $p \in \mathcal{G}$, set $p.\texttt{state} \leftarrow \texttt{far}$ and $U(p) \leftarrow \infty$.
2. For each $p \in \texttt{bd}$, set $p.\texttt{state} \leftarrow \texttt{trial}$, and set $U(p)$ to a user-defined value.
3. While there are `trial` nodes left in $\mathcal{G}$:
   a) Let p_{new} be the `trial` node in `front` with the smallest value $U(p_{\text{new}})$.
   b) Set $p_{\text{new}}.\texttt{state} \leftarrow \texttt{valid}$ and remove p_{new} from `front`.
   c) For each $\hat{p} \in \texttt{nb}(p_{\text{new}})$, set $\hat{p}.\texttt{state} \leftarrow \texttt{trial}$ if $\hat{p}.\texttt{state} = \texttt{far}$.
   d) For each $\hat{p} \in \texttt{nb}(p_{\text{new}})$ such that $\hat{p}.\texttt{state} = \texttt{trial}$, update $\hat{U} = U(\hat{p})$ and merge $\hat{p}$ into `front`.

Specifying how item 3d is to be performed is the crux of developing a Dijkstra-

like algorithm and is left intentionally vague here. This step involves indicating how nodes in `nb`($\hat{p}$) are used to compute $\hat{U}$, and how they are organized into the `front` data structure. The FMM uses an upwind finite difference scheme where only `valid` nodes are used to compute $\hat{U}$, and where nodes on the front are sorted using an array-based heap implementing a priority queue [118]. As an example, Tsitsiklis's algorithm combines nodes in `valid` into sets whose convex hulls approximate the surface of the expanding wavefront and then solves local functional minimization problems. The method presented here is more similar to Tsitsiklis's algorithm (see figure 2.3). For specific details, a general reference should be consulted [118].

In addition to item 3d, algorithm 1 is generic in the following ways:

- As we mentioned before, there are different ways of initializing the boundary data `bd` if only off-grid boundary data is provided [35].

- How we keep track of the node with the smallest value is variable: most frequently, as in Dijkstra's algorithm, a heap storing pointers to the nodes is used, leading to $O(N^n \log N)$ update operations overall, where N^n is the number of nodes. In fact, there are $O(N^n)$ variations using Dial's algorithm (a bucketed version of Dijkstra's algorithm), but these have not been used as extensively as Dijkstra-like algorithms [134, 70, 141].

- The arrangement of the nodes into a grid or otherwise varies, as do the neighborhoods of each node. This affects the update procedure. A regular grid is simple to deal with, but Dijkstra-like methods have been extended to manifolds and unstructured meshes, where the situation is more involved [72, 119, 24].

Other problems can be solved using Dijkstra-like algorithms: the static Hamilton-Jacobi equation, an anisotropic generalization of the eikonal equation, can be solved using the ordered upwind method [120] or other recently introduced methods [87, 86]. The quasipotential of a nongradient stochastic differential equation can also be

computed using the ordered line integral method, although the considerations are more involved [41, 40, 140].

The fast marching method [118] solves a discretized eikonal equation (eq. 1.4) in an upwind fashion. Throughout, we distinguish between the exact solution u and the numerical solution U, where $\hat{U}$ will always denote the current value to be computed; likewise, any quantity with a hat ($\hat{}$) will denote a quantity evaluated at the node being updated. The ordered line integral method locally and approximately minimizes the minimum action integral of (1.4):

$$\hat{u} = \min_{\alpha} \left\{ u_0 + \int_{\alpha} s(x) dl \right\}, \tag{2.1}$$

where α is a ray parametrized by arc length, $\hat{x}$ is a target point, $\hat{u} = u(\hat{x})$, and $u_0 = u(\alpha(0))$. By constrast, Lagrangian methods (i.e., raytracing methods) trace a bundle of rays from a common locus by integrating Hamilton's equations for the eikonal equation for different initial conditions.

In this section, we describe how we discretize and minimize an approximation of eq. 2.1. To compute $\hat{U} = U(\hat{p})$ in item 3d of algorithm 1, we need to approximately minimize several instances of an approximation to eq. 2.1; details of this procedure are discussed in section 2.10. In this section, we focus on a single instance of the discretized version of eq. 2.1. We present our notation, derive prelimary results, and describe the quadrature rules `mp0`, `mp1`, and `rhr`. We also show how the functional minimization problem can be solved exactly using a QR decomposition for the `rhr` and `mp0` rules. Finally, we present theoretical results justifying our approach.

2.3 Approximating the action functional

In this section we describe how we approximate eq. 2.1 and reformulate it as a constrained optimization problem which we solve to update the `trial` nodes surrounding a node p_{new} which has just become `valid`.

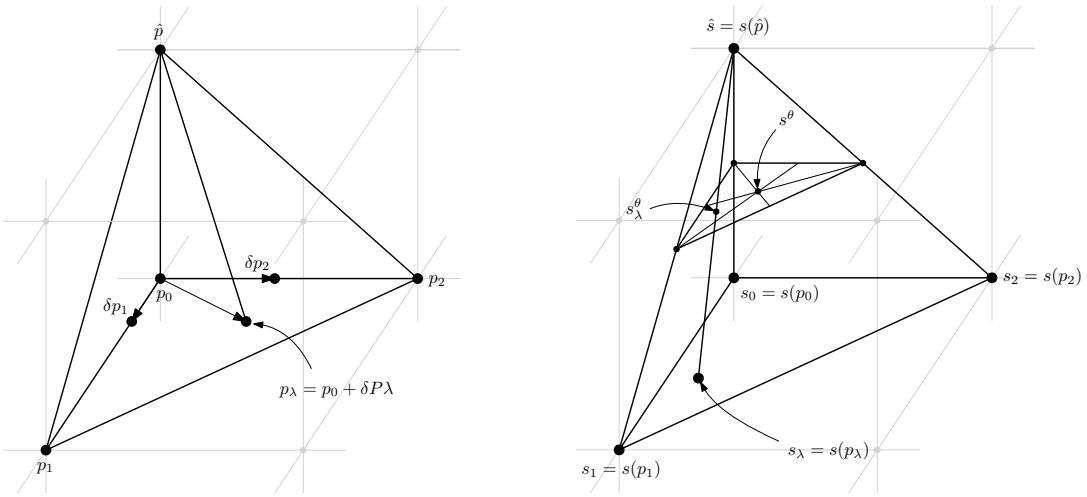


Figure 2.4: *Overview of a tetrahedron update, showing the notation in section 2.2.* Left: a point being updated, $\hat{p}$, which is identified with the origin, and three neighboring points p_0, p_1, and p_2 that are assumed to be `valid`. The grid $\mathcal{G}$, which contains other points in the discretized domain, is sketched in light grey. The domain of the minimization problem eq. 2.12 is the convex hull of p_0, p_1, and p_2. The path minimizing eq. 2.1 is assumed to be the line segment connecting $\hat{p}$ and p_λ. Not pictured is the newly `valid` point p_{new}, although it is assumed that p_{new} equals one of p_0, p_1, or p_2. Right: the same update tetrahedron, but this time with quantities related to the slowness function depicted.

First, we assume that $\Omega \subseteq \mathbb{R}^n$, where $n = 2, 3$. The methods presented here work for general n. We refer to each update as a "simplex update", since for a dimension n, we need to consider updates of dimensions d where $0 \leq d < n$. For $n = 3$, we have line updates ($d = 0$), triangle updates ($d = 1$), and tetrahedron updates ($d = 2$). So, d refers to the dimension of the base of each simplex, which is the dimension of the domain of the optimization problem that we will formulate.

We assume that each update simplex is nondegenerate and that the convex hull of the update point $\hat{p}$ and $d+1$ points $p_0, \ldots, p_d \in \texttt{nb}(\hat{p})$. Since we assume that our grid $\mathcal{G}$ is uniform and rectilinear, we scale and translate $\mathcal{G}$ so that $\hat{p} = 0$ and $\|p_i\|_\infty = 1$ for $i = 0, \ldots, d$. *Throughout the rest of this chapter, we always shift the node $\hat{p}$ to the origin, as this simplifies our calculations.*

Approximating the integration path with a straight line segment. To approximately minimize eq. 2.1 we assume that the minimizing path is a straight line segment connecting $\hat{p}$ and a point in the convex hull of $\{p_0, \ldots, p_d\}$, and numerically approximate the action over this integral path using quadrature. We discuss each part of this approximation in turn. First, some notation.

We parametrize the "base of the update simplex" (the convex hull of $p_0, \ldots, p_d$) over the set:

$$\Delta^d = \left\{ \lambda_i \geq 0 \text{ for } i = 1, \ldots, d \text{ and } \sum_{i=1}^{d} \lambda_i \leq 1 \right\}. \tag{2.2}$$

If we let $\lambda_0 = 1 - \sum_{i=1}^{d} \lambda_i$, then $(\lambda_0, \ldots, \lambda_d)$ is a vector of convex coefficients. We let $\delta p_i = p_i - p_0$ and define:

$$\delta P = \begin{bmatrix} \delta p_1 & \cdots & \delta p_d \end{bmatrix} \in \mathbb{R}^{n \times d}. \tag{2.3}$$

We write a point in the base of the update simplex as:

$$p_\lambda = p_0 + \sum_{i=1}^{d} (p_i - p_0)\lambda_i = p_0 + \sum_{i=1}^{d} \delta p_i \lambda_i = p_0 + \delta P \lambda. \tag{2.4}$$

We will use the "δ" notation for differences and λ as a subscript to denote convex combinations in other contexts, as well. E.g., $\delta U_i = U_i - U_0 = U(x_i) - U(x_0)$ and $U_\lambda = U_0 + \delta U^\top \lambda$. Likewise, $\delta s_i = s_i - s_0 = s(x_i) - s(x_0)$. By an abuse of notation, we will think of, e.g., s_i and $s(x_i)$ in the context of an update as "the same", preferring the notation s_i.

Quadrature rules. We consider a righthand rule (`rhr`), a simplified midpoint rule (`mp0`), and a midpoint rule (`mp1`). Recall that $\hat{s} = s(\hat{x})$. The cost functions being minimized in eq. 2.1 are:

$$F_{\text{rhr}}(\lambda) = U_\lambda + \hat{s} h \|p_\lambda\|, \tag{2.5}$$

$$F_{\text{mp0}}(\lambda) = U_\lambda + \left(\frac{\hat{s} + \frac{1}{d+1} \sum_{i=0}^{d} s_i}{2} \right) h \|p_\lambda\|, \tag{2.6}$$

$$F_{\text{mp1}}(\lambda) = U_\lambda + \left(\frac{\hat{s} + s_\lambda}{2} \right) h \|p_\lambda\|. \tag{2.7}$$

The difference in the quadrature rules, of course, lies in how we incorporate the slowness s. For F_{rhr}, we evaluate s at the righthand side of the integral, yielding $\hat{s}$. For F_{mp1}, we evaluate s at the midpoint of the integral, approximating s linearly with the convex combination s_λ on the base of the simplex. Finally, for F_{mp0}, we approximate s_λ itself with the arithmetic mean of the s_i's. It will turn out that F_{mp0} will lead to an inconsistent numerical scheme unless extra care is taken, which is discussed in the rest of the work.

Note that in the above we use s_λ and not $s(p_\lambda)$ because we do not want to assume that we have access to a continuous functional form for s; in most cases, we assume that s will be provided as gridded data, and that interpolation will be used to approximate s off-grid.

More general quadrature rules. The quadrature rules above are specializations of the following more general quadrature rules. With θ such that $0 \leq \theta \leq 1$, we define:

$$F_0(\lambda) = U_\lambda + \left[(1-\theta)\hat{s} + \frac{\theta}{d+1}\sum_{i=0}^{d} s_i\right] h\|p_\lambda\|, \tag{2.8}$$

$$F_1(\lambda) = U_\lambda + \Big[(1-\theta)\hat{s} + \theta s_\lambda\Big] h\|p_\lambda\|. \tag{2.9}$$

Then, $F_{\text{rhr}} = F_0 = F_1$ with $\theta = 0$, $F_{\text{mp0}} = F_0$ with $\theta = \frac{1}{2}$, and $F_{\text{mp1}} = F_1$ with $\theta = \frac{1}{2}$. To simplify notation in our proofs, we also define:

$$s^\theta = (1-\theta) + \frac{\theta}{d+1}\sum_{i=0}^{d} s_i, \qquad s_\lambda^\theta = (1-\theta)\hat{s} + \theta s_\lambda. \tag{2.10}$$

Then, the θ-rules can be written more compactly as:

$$F_0(\lambda) = U_\lambda + s^\theta h\|p_\lambda\|, \qquad F_1(\lambda) = U_\lambda + s_\lambda^\theta h\|p_\lambda\|. \tag{2.11}$$

We introduce this more general "θ-rule" for two reasons:

- This is a natural geometric generalization, and we wish to contextualize our results properly.

- Our proofs are written in terms of the θ-rules, which allows us to provide proofs for F_0 and proofs for F_1, instead of separate proofs for each of F_{rhr}, F_{mp0}, and F_{mp1}. These proofs apply to other θ-rules not considered here; e.g., setting $\theta = 1$ or choosing θ adaptively.

2.4 The minimization problem

With F_0 and F_1 so defined, the minimization problem which approximates eq. 2.1 is:

$$\hat{U} = \min_{\lambda \in \Delta^d} F(\lambda), \tag{2.12}$$

where $F = F_{\text{rhr}}, F_{\text{mp0}}$, or F_{mp1}. This is a nonlinear, constrained optimization problem with linear inequality constraints and no equality constraints. We require the gradient and Hessian of F_0 and F_1 for our algorithms and analysis. These are easy to compute, but we have found a particular form for them to be convenient for both implementation and analysis.

In what follows, we will use the notation:

$$\text{proj}_p = \frac{pp^\top}{p^\top p}, \qquad \text{proj}_p^\perp = I - \text{proj}_p = I - \frac{pp^\top}{p^\top p} \tag{2.13}$$

for orthogonal projection matrices. Here, proj_p projects orthogonally onto $\text{span}(p)$, and $\text{proj}_p^\perp$ onto its orthogonal complement.

Proposition 2. *The gradient and Hessian of $F_0(\lambda)$ are given by:*

$$\nabla F_0(\lambda) = \delta U + s^\theta h \delta P^\top \nu_\lambda, \tag{2.14}$$

$$\nabla^2 F_0(\lambda) = \frac{s^\theta h}{\|p_\lambda\|} \delta P^\top \text{proj}_{p_\lambda}^\perp \delta P, \tag{2.15}$$

where $\nu_\lambda = p_\lambda / \|p_\lambda\|$ is the unit vector in the direction of p_λ.

Proof. For the gradient, we have:

$$\nabla F_0(\lambda) = \delta U + \frac{s^\theta h}{2\|p_\lambda\|} \nabla p_\lambda^\top p_\lambda = \delta U + \frac{s^\theta h}{\|p_\lambda\|} \delta P^\top p_\lambda,$$

since $\nabla p_\lambda^\top p_\lambda = 2\delta P^\top p_\lambda$. For the Hessian:

$$\begin{aligned}\nabla_\lambda^2 F_0(\lambda) &= \nabla\left(\frac{s^\theta h}{\|p_\lambda\|}p_\lambda^\top \delta P\right) = s^\theta h\left(\nabla\frac{1}{\|p_\lambda\|}p_\lambda^\top\delta P + \frac{1}{\|p_\lambda\|}\nabla p_\lambda^\top \delta P\right)\\ &= \frac{s^\theta h}{\|p_\lambda\|}\left(\delta P^\top\delta P - \frac{\delta P^\top p_\lambda p_\lambda^\top \delta P}{p_\lambda^\top p_\lambda}\right) = \frac{s^\theta h}{\|p_\lambda\|}\delta P^\top\left(I - \frac{p_\lambda p_\lambda^\top}{p_\lambda^\top p_\lambda}\right)\delta P,\end{aligned}$$

from which the result follows. □

Proposition 3. *The gradient and Hessian of $F_1(\lambda)$ satisfy:*

$$\nabla F_1(\lambda) = \delta U + \theta h\|p_\lambda\|\delta s + s_\lambda^\theta h\delta P^\top \nu_\lambda, \tag{2.16}$$

$$\nabla^2 F_1(\lambda) = \left\{\delta P^\top \nu_\lambda, \theta h\delta s\right\} + \frac{s_\lambda^\theta h}{\|p_\lambda\|}\delta P^\top \operatorname{proj}_{p_\lambda}^\perp \delta P, \tag{2.17}$$

where $\{a,b\} = ab^\top + ba^\top$ is the anticommutator of two vectors.

Proof. Since $F_1(\lambda) = u_\lambda + hs_\lambda^\theta\|p_\lambda\|$, for the gradient we have:

$$\nabla F_1(\lambda) = \delta U + h\left(\theta\|p_\lambda\|\delta s + \frac{s_\lambda^\theta}{2\|p_\lambda\|}\nabla p_\lambda^\top p_\lambda\right) = \delta U + \frac{h}{\|p_\lambda\|}\left(\theta p_\lambda^\top p_\lambda\delta s + s^\theta \delta P^\top p_\lambda\right),$$

and for the Hessian:

$$\begin{aligned}\nabla^2 F_1(\lambda) = \frac{h}{2\|p_\lambda\|}\Bigg(&\theta\Big(\nabla p_\lambda^\top p_\lambda \delta s^\top + \ \delta s(\nabla p_\lambda^\top p_\lambda)^\top\Big) + \\ &s_\lambda^\theta\left(\frac{1}{2p_\lambda^\top p_\lambda}\nabla p_\lambda^\top p_\lambda(\nabla p_\lambda^\top p_\lambda)^\top - \nabla_\lambda^2 p_\lambda^\top p_\lambda\right)\Bigg).\end{aligned}$$

Simplifying this gives us the result. □

Our task is to minimize F_0 and F_1 over the convex set Δ^n; so, we need to determine whether F_0 and F_1 are convex functions. The next two lemmas address this point.

Lemma 1. *Let $p_0,\dots,p_d$ form a nondegenerate simplex (i.e., $p_0,\dots,p_d$ are linearly independent) together with $\hat{p}$ and assume that s is positive. Then, $\nabla^2 F_0$ is positive definite and F_0 is strictly convex.*

Proof. Let $\nu_\lambda = p_\lambda / \|p_\lambda\| \in \mathbb{R}^n$ be the unit vector in the direction of p_λ, and assume that $Q = \begin{bmatrix} \nu_\lambda & U \end{bmatrix} \in \mathbb{R}^{n \times n}$ is orthonormal. Then:

$$\delta P^\top \operatorname{proj}_{p_\lambda}^\perp \delta P = \delta P^\top (I - \nu_\lambda \nu_\lambda^\top) \delta P = \delta P^\top (QQ^\top - \nu_\lambda \nu_\lambda^\top) \delta P = \delta P^\top U U^\top \delta P. \quad (2.18)$$

Hence, $\delta P^\top \operatorname{proj}^\perp \delta P$ is a Gram matrix and positive semidefinite.

Next, since Δ^n is nondegenerate, the vectors p_i for $i = 0, \ldots, n-1$ are linearly independent. Since the ith column of δP is $\delta p_i = p_i - p_0$, we can see that the vector p_0 is not in the range of δP; hence, there is no vector μ such that $\delta P \mu = \alpha p_\lambda$, for any $\alpha \neq 0$. What's more, by definition, $\ker(\operatorname{proj}_{p_\lambda}^\perp) = \langle p_\lambda \rangle$. So, we can see that $\operatorname{proj}_{p_\lambda}^\perp \delta P \mu = 0$ only if $\mu = 0$, from which we can conclude $\delta P^\top \operatorname{proj}_{p_\lambda}^\perp \delta P \succ 0$. Altogether, bearing in mind that $s_{\min}$ is assumed to be positive, we conclude that $\nabla^2 F_0$ is positive definite. □

For F_1, we can only obtain convexity (let alone strict convexity) for h sufficiently small. For large enough h, we will encounter nonconvex updates. To obtain convexity, we need to stipulate that the slowness function s is Lipschitz continuous on Ω with a Lipschitz constant that is independent of h. In practice, we have not found this to be a particularly stringent restriction.

Lemma 2. *In the setting of lemma 1, additionally assume that s is Lipschitz continuous with Lipschitz constant $K \leq C$ on Ω, for some constant $C > 0$ independent of h. Then, $\nabla^2 F_1$ is positive definite (hence, F_1 is strictly convex) for h small enough.*

Proof. To show that $\nabla^2 F_1$ is positive definite for h small enough, note from eq. 2.17 that $\nabla^2 F_1 = A + B$, where A is positive definite and B is small relative to A and indefinite. To use this fact, note that since $\delta P^\top \operatorname{proj}_\lambda^\perp \delta P$ is symmetric positive definite, it has an eigenvalue decomposition $Q \Lambda Q^\top$ where $\Lambda_{ii} > 0$ for all i. Since $\delta P^\top \operatorname{proj}_\lambda^\perp \delta P$ doesn't depend on h, for a fixed set of vectors $p_0, \ldots, p_n$, its eigenvalues are constant

with respect to h. So, defining:

$$A = \frac{s_\lambda^\theta h}{\|p_\lambda\|}\delta P^\top \operatorname{proj}_\lambda^\perp \delta P = Q\left(\frac{s_\lambda^\theta h}{\|p_\lambda\|}\Lambda\right)Q^\top \tag{2.19}$$

we can expect this matrix's eigenvalues to be $\Theta(h)$; in particular, $\lambda_{\min} \geq Ch$ for some constant C, provided that $s > s_{\min} > 0$, as assumed. This gives us a bound for the positive definite part of ∇F_1^2.

The perturbation $B = \left\{\delta P^\top \nu_\lambda, \theta h \delta s\right\}$ is indefinite. Since $\|\delta s\| = O(h)$, we find that:

$$|\lambda_{\max}(B)| = \left\|\left\{\delta P^\top \nu_\lambda, \theta h \delta s\right\}\right\|_2 \leq \theta h\sqrt{n}\left\|\left\{\delta P^\top \nu_\lambda, \delta s\right\}\right\|_\infty = O(h^2), \tag{2.20}$$

where we use the fact that the Lipschitz constant of s is $K \leq C$, so that:

$$|\delta s_i| = |s_i - s_0| \leq K|x_i - x_0| \leq Kh\sqrt{n} \leq Ch\sqrt{n}, \tag{2.21}$$

for each i. Letting $z \neq 0$, we compute:

$$z^\top \nabla^2 F_1 z = z^\top A z + z^\top B z \geq \lambda_{\min}(A) z^\top z + z^\top B z \geq Chz^\top z + z^\top B z. \tag{2.22}$$

Now, since $\left|z^\top B z\right| \leq |\lambda_{\max}(B)|\, z^\top z \leq Dh^2 z^\top z$, where D is some positive constant, we can see that for h small enough, it must be the case that $Chz^\top z + z^\top B z > 0$; i.e., that $\nabla^2 F_1$ is positive definite; consequently, F_1 is strictly convex in this case. □

We have found that all `mp1` updates become strictly convex problems rapidly as $h \to 0$. The reason for this is discussed at the end of section 2.8.

2.5 Validation of `mp0`

If we use F_{mp0} directly, then we run into a situation where the cost function F_{mp0} is not continuous between the bases of adjacent update simplices (see fig. 2.5). We require F to be continuous across simplex boundaries to avoid an inconsistent or divergent solver. Now, if we first use F_{mp0} to compute the minimizer λ_0^* of eq. 2.12

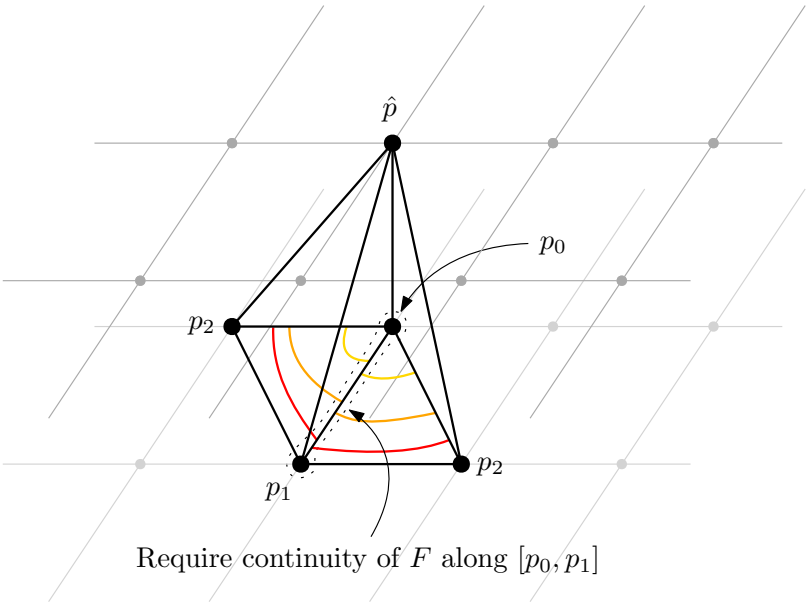


Figure 2.5: *A problem with* `mp0` *for which we provide a simple solution.* The cost function F must be continuous across the boundaries of adjacent update simplexes, otherwise an inconsistent solver can come about. Here, the colorered level sets depict the discontinuity of F_{mp0} across the bases of the update simplexes. In this case, two adjacent update simplexes share a common boundary on their base, shown here as the line segment $[p_0, p_1]$. Two layers of surrounding grid points from $\mathcal{G}$ are shown. The point $\hat{p}$ is in the top layer and the points p_0, p_1, and p_2 are in the bottom layer.

with $F = F_{\text{mp0}}$, and then set $\hat{U} = F_{\text{mp1}}(\lambda_0^*)$, we will recover continuity, and indeed, as we will show, the scheme is convergent. The motivation this is that eq. 2.12 can be solved exactly for the θ-rule F_0 using a QR decomposition instead of an iterative solver, making it very cheap. In the next section, we will show how this can be done.

Let λ_0^* and λ_1^*, denote the optima of eq. 2.12, where $F = F_0$ and $F = F_1$ (the general θ-rules), respectively. We could imagine using Newton's method to minimize F_1, starting from λ_0^* (to be clear, this is not the approach we will ultimately take numerically). This would allow us to use the convergence theory of Newton's method to bound the distance between λ_0^* and λ_1^*, thereby bounding the error incurred by using `mp0` instead of `mp1` to find the minimizing argument of eq. 2.12. We follow this idea now.

We first establish some technical lemmas that we will use to validate the use of `mp0`.

Lemmas 3, 4, and 5 set up the conditions for theorem 1 of Stoer and Bulirsch [127], from which theorem 2 readily follows.

Lemma 3. *There exists* $\beta = O(h^{-1})$ *s.t.* $\|\nabla^2 F_1(\lambda)^{-1}\| \leq \beta$ *for all* $\lambda \in \Delta^n$.

Proof. To simplify eq. 2.17, we temporarily define:

$$A = \frac{s_\lambda^\theta h}{\|p_\lambda\|}\delta P^\top \operatorname{proj}_\lambda^\perp \delta P \text{ and } B = \frac{\theta h}{\|p_\lambda\|}\left\{\delta P^\top p_\lambda, \delta s\right\}. \tag{2.23}$$

Observe that $\|A\| = O(h)$ and $\|B\| = O(h^2)$, since $\|\delta s\| = O(h)$ and since all other factors involved in A and B (excluding h itself) are independent of h. Hence:

$$\left\|A^{-1}B\right\| = \frac{\theta}{s_\lambda^\theta}\left\|\left(\delta P^\top \operatorname{proj}_\lambda^\perp \delta P\right)^{-1}\left\{\delta P^\top p_\lambda, \delta s\right\}\right\| = O(h), \tag{2.24}$$

since $\|\delta s\| = O(h)$. Hence, $\|A^{-1}B\| < 1$ for h small enough, and we can Taylor expand:

$$\begin{aligned}\nabla^2 F_1(\lambda)^{-1} &= (A+B)^{-1} = (I + A^{-1}B)^{-1}A^{-1} \\ &= \left(I - A^{-1}B + (A^{-1}B)^2 - \cdots\right)A^{-1} \\ &= A^{-1} - A^{-1}BA^{-1} + (A^{-1}B)^2A^{-1} - \cdots,\end{aligned} \tag{2.25}$$

which implies $\|\nabla^2 F_1(\lambda)^{-1}\| = O(h^{-1})$. Note that when we Taylor expand, $\|A^{-1}B\| = O(h)$, so that $\|A^{-1}B\| < 1$ for h small enough. To define β, let:

$$\beta = \max_{\lambda\in\Delta^n}\left\|\nabla^2 F_1(\lambda)^{-1}\right\| = O(h^{-1}), \tag{2.26}$$

completing the proof. □

Lemma 4. *There exists* $\alpha = O(h)$ *s.t.* $\|\nabla^2 F_1(\lambda_0^*)^{-1}\nabla F_1(\lambda_0^*)\| \leq \alpha$.

Proof. From lemma 3 we have $\|F_1(\lambda_0^*)^{-1}\| = O(h^{-1})$, so to establish the result we only need to show that $\|\nabla F_1(\lambda_0^*)\| = O(h^2)$. To this end, let $\underline{\lambda} = (n+1)^{-1}\mathbf{1}_{n\times 1}$ (i.e., the centroid of Δ^n, where s^θ is evaluated). Then, recalling figure 2.4, $s_\lambda^\theta = s^\theta + \delta s^\top(\lambda - \underline{\lambda})$ so that, for a general λ:

$$\begin{aligned}\nabla F_1(\lambda) &= \|p_\lambda\|h\delta s + \delta U + \frac{s^\theta + \delta s^\top(\lambda - \underline{\lambda})}{\|p_\lambda\|}h\delta P^\top p_\lambda \\ &= \|p_\lambda\|h\delta s + \nabla F_0(\lambda) + \frac{\delta s^\top(\lambda - \underline{\lambda})}{\|p_\lambda\|}h\delta P^\top p_\lambda.\end{aligned} \tag{2.27}$$

Since $\nabla F_0(\lambda_0^*) = 0$ by optimality, we can conclude using eq. 2.27 and $\|\delta s\| = O(h)$ that:

$$\|\nabla F_1(\lambda_0^*)\| = h \left\| \|p_{\lambda_0^*}\| \delta s + \frac{\delta s^\top (\lambda - \underline{\lambda})}{\|p_{\lambda_0^*}\|} \delta P^\top p_\lambda \right\| = O(h^2), \tag{2.28}$$

which proves the result. □

Lemma 5. *The Hessian $\nabla^2 F_1$ is Lipschitz continuous with $O(h)$ Lipschitz constant. That is, there is some constant $\gamma = O(h)$ so that for two points λ and λ':*

$$\left\| \nabla^2 F_1(\lambda) - \nabla^2 F_1(\lambda') \right\| \leq \gamma \left\| \lambda - \lambda' \right\|.$$

Proof. If we restrict our attention to Δ^n, we see that $\|p_\lambda\|^{-1} \delta P^\top \operatorname{proj}_\lambda^\perp \delta P$ is Lipschitz continuous function of λ with $O(1)$ Lipschitz constant and $\theta\{\delta P^\top p_\lambda, \delta s\}/\|p_\lambda\|$ is Lipschitz continuous with $O(h)$ Lipschitz constant since $\|\delta s\| = O(h)$. Then, since s_λ^θ is $O(1)$ Lipschitz, it follows that:

$$A(\lambda) = \tfrac{s_\lambda^\theta h}{\|p_\lambda\|} \delta P^\top \operatorname{proj}_\lambda^\perp \delta P \tag{2.29}$$

has a Lipschitz constant that is $O(h)$ for $\lambda \in \Delta^n$, using the notation of lemma 3. Likewise,

$$B(\lambda) = \tfrac{\theta h}{\|p_\lambda\|} \left\{ \delta P^\top p_\lambda, \delta s \right\} = O(h^2), \tag{2.30}$$

since it is a sum of two terms involving products of h and δs. Since $\nabla^2 F_1(\lambda) = A(\lambda) + B(\lambda)$, we can see immediately that it is also Lipschitz on Δ^n with a constant that is $O(h)$. □

For our proof of Theorem 2, we need to following theorem version of Kantorovich's theorem on the convergence of Newton's method.

Theorem 1 (Theorem 5.3.2, Stoer and Bulirsch)**.** *Let $C \subseteq \mathbb{R}^n$ be an open set, let C_0 be a convex set with $\overline{C}_0 \subseteq C$, and let $f : C \to \mathbb{R}^n$ be differentiable for $x \in C_0$ and continuous for $x \in C$. For $x_0 \in C_0$, let r, α, β, γ satisfy $S_r(x_0) = \{x : \|x - x_0\| < r\} \subseteq C_0$, $\mu = \alpha\beta\gamma < 2$, $r = \alpha(1-\mu)^{-1}$, and let f satisfy:*

(a) for all $x, y \in C_0$, $\|Df(x) - Df(y)\| \leq \gamma \|x - y\|$,

(b) for all $x \in C_0$, $(Df(x))^{-1}$ *exists and satisfies* $\|(Df(x))^{-1}\| \leq \beta$,

(c) and $\|(Df(x_0))^{-1} f(x_0)\| \leq \alpha$.

Then, beginning at x_0*, each iterate:*

$$x_{k+1} = x_k - Df(x_k)^{-1} f(x_k), \qquad k = 0, 1, \ldots, \tag{2.31}$$

is well-defined and satisfies $\|x_k - x_0\| < r$ *for all* $k \geq 0$*. Furthermore,* $\lim_{k\to\infty} x_k = \xi$ *exists and satisfies* $\|\xi - x_0\| \leq r$ *and* $f(\xi) = 0$.

Theorem 2. *Using lemma 2, let* h *be sufficiently small so that* F_1 *is strictly convex. Then, the error* $\delta\lambda^* = \lambda_1^* - \lambda_0^*$ *satisfies* $\|\delta\lambda^*\| = O(h)$*. Further, if we let* $\lambda_0 = \lambda_0^*$ *in the following Newton iteration:*

$$\lambda_{k+1} \leftarrow \lambda_k - \nabla^2 F_1(\lambda_k)^{-1} \nabla F_1(\lambda_k), \qquad k = 0, 1, \ldots, \tag{2.32}$$

then this iteration is well-defined, and converges quadratically to λ_1^**. This immediately implies that the error incurred by* `mp0` *is* $O(h^3)$ *per update compared to* `mp1`*; i.e.:*

$$|F_1(\lambda_1^*) - F_1(\lambda_0^*)| = O(h^3). \tag{2.33}$$

Proof. Our proof of theorem 2 relies on the following theorem on the convergence of Newton's method, which we present for convenience.

For our situation, Theorem 5.3.2 of Stoer and Bulirsch [127] indicates that if:

$$\|\nabla F_1(\lambda)^{-1}\| \leq \beta, \text{where } \beta = O(h^{-1}), \tag{2.34}$$

$$\|\nabla F_1(\lambda_0^*)^{-1} \nabla F_1(\lambda_0^*)\| \leq \alpha, \text{where } \alpha = O(h), \text{ and} \tag{2.35}$$

$$\|\nabla F_1(\lambda) - \nabla F_1(\lambda')\| \leq \gamma \|\lambda - \lambda'\| \text{ for each } \lambda, \lambda' \in \Delta^n, \text{where } \gamma = O(h), \tag{2.36}$$

then with $\lambda_0 = \lambda_0^*$, the iteration eq. 2.32 is well-defined, with each iterate satisfying $\|\lambda_k - \lambda_0\| \leq r$, where $r = \alpha/(1 - \alpha\beta\gamma/2)$. Additionally, the limit of this iteration

exists, and the iteration converges to it quadratically; we note that since F_1 is strictly convex for h small enough, the limit of the iteration must be λ_1^*, so the theorem also gives us $\|\delta\lambda^*\| = \|\lambda_1^* - \lambda_0^*\| \leq r$.

Now, we note that items 2.34, 2.35, and 2.36 correspond exactly to lemma 3, 4, and 5, respectively which gave us values for α, β, and γ. All that remains is to compute r. Since the preceding lemmas imply $\alpha\beta\gamma = O(h)$, hence $\alpha\beta\gamma/2 < 1$ for h small enough. We have:

$$r = \frac{\alpha}{1 - \frac{\alpha\beta\gamma}{2}} = \alpha\left(1 + \frac{\alpha\beta\gamma}{2} + \frac{\alpha^2\beta^2\gamma^2}{4} + \cdots\right) = O(h), \tag{2.37}$$

so that $\|\delta\lambda^*\| = O(h)$, and the result follows.

To obtain the $O(h^3)$ error bound, from theorem 2, we have $\|\delta\lambda^*\| = O(h)$. Then, Taylor expanding $F_1(\lambda_0^*)$, we get:

$$F_1(\lambda_0^*) = F_1(\lambda_1^* - \delta\lambda^*) = F_1(\lambda_1^*) - \nabla F_1(\lambda_1^*)^\top \delta\lambda^* + \frac{1}{2}\delta\lambda^* \nabla F_1^2(\lambda_1^*)\delta\lambda^* + R,$$

where $|R| = O(\|\delta\lambda^*\|^3)$. Since λ_1^* is optimum, $\nabla F_1(\lambda_1^*) = 0$. Hence:

$$|F_1(\lambda_1^*) - F_1(\lambda_0^*)| \leq \frac{1}{2}\left\|\nabla F_1^2(\lambda_1^*)\right\| \|\delta\lambda^*\|^2 + O(\|\delta\lambda^*\|^3) = O(h^3),$$

which proves the result. □

We will show in the next section how λ_0^* can be computed directly using a QR decomposition and without using an iterative solver.

We can provide some intuition for why this bound is satisfactory. If we assume that our domain is spanned along a diameter by $O(N)$ nodes, and that $h \sim N^{-1}$, then we can anticipate $O(N)$ downwind updates, starting from `bd` and extending to the boundary of $\mathcal{G}$ in any direction. Accumulating the error over these nodes, we can expect the maximum pointwise error between a solution to eq. 1.4 computed by using `mp0` and `mp1` to be $O(h^2)$, which is dominated by the $O(h)$ discretization error coming from the linear convergence of the method itself. Hence, using `mp0` instead of `mp1` only to find the parameter λ, and then evaluate $\hat{U}$ using F_1, should introduce no significant extra error.

2.6 Exact minimization using a QR decomposition

Since F_0 is strictly convex, $\nabla F_0(\lambda) = 0$ is sufficient for the optimality of λ, ignoring the constraint $\lambda \in \Delta^d$. The unconstrained system of nonlinear equations defined by $\nabla F_0(\lambda) = 0$ can be solved exactly without an iterative solver. We can compute the solution using the reduced QR decomposition of δP and by considering the problem's geometry (see also Figure 2.6). This is captured in the following theorem. We will discuss how to use this theorem efficiently in a solver in section 2.10.

Theorem 3. *Let $\delta P = QR$ be the reduced QR decomposition of δP; i.e., where $Q \in \mathbb{R}^{n\times d}, R \in \mathbb{R}^{d\times d}, Q^\top Q = I_d$, and with R upper triangular. For s^θ, h, and U fixed, if $\lambda^* = \operatorname{argmin}_{\lambda\in\mathbb{R}^n} F_0(\lambda)$, then:*

$$\|p_{\lambda^*}\| = \sqrt{\frac{p_0^\top (I - QQ^\top) p_0}{1 - \left\|R^{-\top}\frac{\delta U}{s^\theta h}\right\|^2}}, \tag{2.38}$$

$$\lambda^* = -R^{-1}\left(Q^\top p_0 + \|p_{\lambda^*}\| R^{-\top}\frac{\delta U}{s^\theta h}\right), \tag{2.39}$$

$$\hat{U} = U_0 + \frac{s^\theta h}{\|p_{\lambda^*}\|} p_0^\top p_{\lambda^*}. \tag{2.40}$$

Proof of theorem 3. We proceed by reasoning geometrically; figure 2.6 depicts the geometric setup. First, letting $\delta P = QR$ be the reduced QR decomposition of δP, and writing $\nu_{\lambda^*} = p_{\lambda^*}/\|p_{\lambda^*}\|$, we note that since:

$$\nabla F_0(\lambda^*) = \delta U + s^\theta h \delta P^\top \nu_{\lambda^*} = 0, \tag{2.41}$$

the optimum λ^* satisfies:

$$-R^{-\top}\frac{\delta U}{s^\theta h} = Q^\top \nu_{\lambda^*} \tag{2.42}$$

Let $\text{proj}_{\delta P} = QQ^\top$ denote the orthogonal projector onto range(δP), and $\text{proj}^\perp_{\delta P} = I - QQ^\top$ the projector onto its orthogonal complement. We can try to write p_{λ^*} by

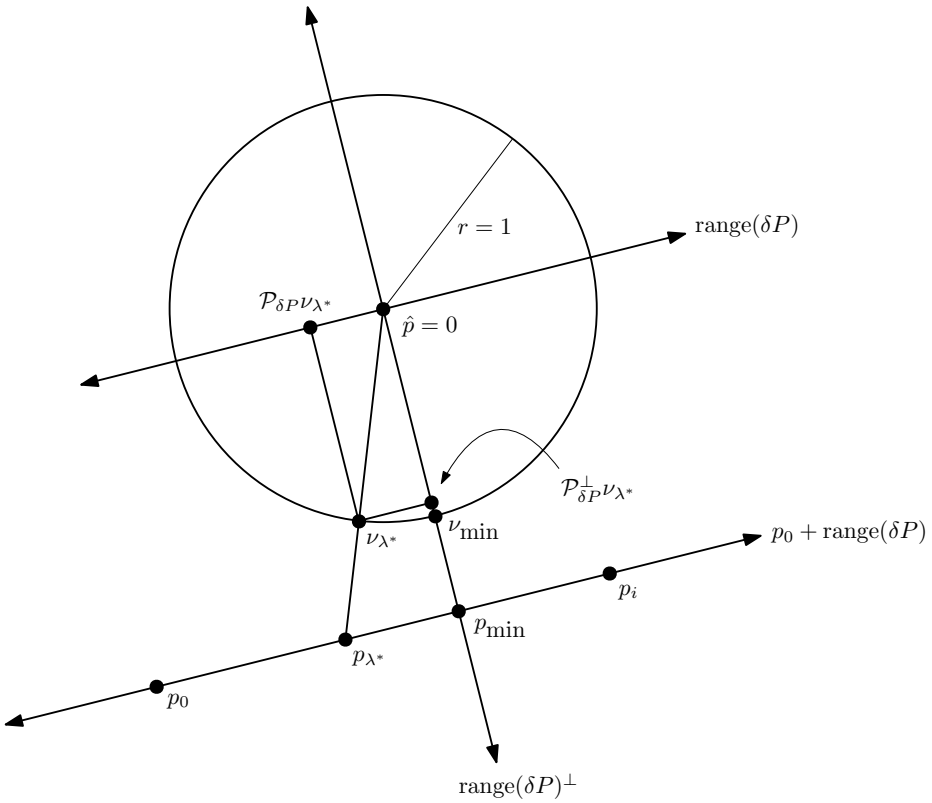


Figure 2.6: A schematic depiction of the proof of theorem 3.

splitting it into a component that lies in range(δP) and one that lies in range(δP)$^{\perp}$. Letting $p_{\min}$ be the point in $p_0 + \text{range}(\delta P)$ with the smallest 2-norm, we write:

$$p_{\lambda^*} = (p_{\lambda^*} - p_{\min}) + p_{\min}, \tag{2.43}$$

where $p_{\lambda^*} - p_{\min} \in \text{range}(\delta P)$ and $p_{\min} \in \text{range}(\delta P)^{\perp}$. The vector $p_{\min}$ corresponds to $p_{\lambda_{\min}}$ where $\lambda_{\min}$ satisfies:

$$0 = \delta P^{\top}(\delta P \lambda_{\min} + p_0) = R^{\top} R \lambda_{\min} + R^{\top} Q^{\top} p_0, \tag{2.44}$$

hence $\lambda_{\min} = -R^{-1}Q^{\top}p_0$, giving us:

$$p_{\min} = p_0 + \delta P \lambda_{\min} = \text{proj}^{\perp}_{\delta P}\, p_0. \tag{2.45}$$

This vector is easily obtained. For $p_{\lambda^*} - p_{\min}$, we note that $\text{proj}_{\delta P}\, \nu_{\lambda^*}$ is proportional to $p_{\lambda^*} - p_{\min}$, suggesting that we determine the ratio α satisfying $p_{\lambda^*} - p_{\min} = \alpha\, \text{proj}_{\delta P}\, \nu_{\lambda^*}$.

In particular, from the similarity of the triangles $(\hat{p}, \nu_{\lambda^*}, \text{proj}^{\perp}_{\delta P}\, \nu_{\lambda^*})$ and $(\hat{p}, p_{\lambda^*}, p_{\min})$ in figure 2.6, we have, using eqs. 2.42 and 2.45:

$$\alpha = \frac{\|p_{\min}\|}{\left\|\text{proj}^{\perp}_{\delta P}\, \nu_{\lambda^*}\right\|} = \sqrt{\frac{p_0^{\top}\, \text{proj}^{\perp}_{\delta P}\, p_0}{1 - \|Q^{\top}\nu_{\lambda^*}\|^2}} = \sqrt{\frac{p_0^{\top}\, \text{proj}^{\perp}_{\delta P}\, p_0}{1 - \left\|R^{-\top}\frac{\delta U}{s^{\theta}h}\right\|^2}}. \tag{2.46}$$

At the same time, since:

$$\nu_{\lambda^*}^{\top}\, \text{proj}^{\perp}_{\delta P}\, \nu_{\lambda^*} = \frac{\left(\text{proj}^{\perp}_{\delta P}\, p_{\lambda^*}\right)^{\top}\left(\text{proj}^{\perp}_{\delta P}\, p_{\lambda^*}\right)}{\|p_{\lambda_*}\|^2} = \frac{p_{\min}^{\top} p_{\min}}{\|p_{\lambda^*}\|^2} = \frac{p_0^{\top}\, \text{proj}^{\perp}_{\delta P}\, p_0}{\|p_{\lambda^*}\|^2} \tag{2.47}$$

we can conclude that:

$$\|p_{\lambda^*}\| = \alpha = \sqrt{\frac{p_0^{\top}\, \text{proj}^{\perp}_{\delta P}\, p_0}{1 - \left\|R^{-\top}\frac{\delta U}{s^{\theta}h}\right\|^2}}, \tag{2.48}$$

giving us eq. 2.38, proving the first part of theorem 3.

Next, combining eqs. 2.42, 2.43, 2.45, and 2.46, we get:

$$p_{\lambda^*} = \text{proj}^{\perp}_{\delta P}\, p_0 - \sqrt{\frac{p_0^{\top}\, \text{proj}^{\perp}_{\delta P}\, p_0}{1 - \left\|R^{-\top}\frac{\delta U}{s^{\theta}h}\right\|^2}} Q R^{-\top}\frac{\delta U}{s^{\theta}h}. \tag{2.49}$$

This expression for p_{λ^*} can be computed from our problem data and δP. Now, note that $p_{\lambda^*} = p_0 + \delta P \lambda^*$ implies:

$$\lambda^* = R^{-1}Q^{\top}(p_{\lambda^*} - p_0). \tag{2.50}$$

Substituting eq. 2.49 into eq. 2.50, we obtain eq. 2.39 after making appropriate cancellations, establishing the second part of theorem 3.

To establish eq. 2.40, we note that by optimality of λ^*, our expression for ∇F_0 (eq. 2.14 of proposition 2) gives:

$$\delta U = -s^{\theta}h\frac{\delta P^{\top} p_{\lambda^*}}{\|p_{\lambda^*}\|}. \tag{2.51}$$

This lets us write:

$$\delta U^{\top}\lambda^* = -\frac{s^{\theta}h}{\|p_{\lambda^*}\|}p_{\lambda^*}^{\top}\delta P^{\top}\lambda^* = \frac{s^{\theta}h}{\|p_{\lambda^*}\|}p_{\lambda^*}^{\top}(p_0 - p_{\lambda^*}). \tag{2.52}$$

Combining eq. 2.52 with our definition of F_0 yields:

$$\hat{U} = F_0(\lambda^*) = U_0 + \delta U^{\top}\lambda^* + s^{\theta}h\|p_{\lambda^*}\| = U_0 + \frac{s^{\theta}h}{\|p_{\lambda^*}\|}p_{\lambda^*}^{\top}(p_0 - p_{\lambda^*}) + \frac{s^{\theta}h}{\|p_{\lambda^*}\|}p_{\lambda^*}^{\top}p_{\lambda^*}, \tag{2.53}$$

which gives eq. 2.40, completing the final part of the proof. □

2.7 Equivalence of the upwind finite difference scheme and F_0

If we linearly approximate U near $\hat{p}$, then for $i = 0, \ldots, n-1$, we find that $\hat{U}$ satisfies:

$$U_i - \hat{U} = \nabla \hat{U}^\top p_i. \tag{2.54}$$

This finite difference approximation to eq. 1.4 can be solved exactly and is a known generalization of the upwind finite difference scheme used in the fast marching method on an unstructured mesh [72, 119]. Computing $\hat{U}$ using this approximation is equivalent to solving:

$$\hat{U} = \min_{\lambda \in \Delta^n} F_0(\lambda) \tag{2.55}$$

in a sense made precise by the following theorem. As we have pointed out, this theorem is not new, but we present it here for the sake of continuity and because it dovetails with our other theorems, providing context.

Theorem 4 (Equivalence of upwind finite difference scheme and F_0)**.** *Let $\hat{U}$ by the solution of eq. 2.54 and let $\hat{U}' = \min_{\lambda \in \mathbb{R}^n} F_0(\lambda)$. Then, $\hat{U}$ exists if and only if $\|R^{-\top} \delta U\| \leq s^\theta h$, and can be computed from:*

$$\hat{U} = U_i - p_i^\top Q R^{-\top} \delta U + \|p_{\min}\| \sqrt{(s^\theta h)^2 - \|R^{-\top} \delta U\|^2}, \tag{2.56}$$

where $p_{\min} = (I - QQ^\top) p_i$ (for all $i = 0, \ldots, n-1$—see Figure 2.6. Additionally, the following hold:

1. *The finite difference solution and line integral solution coincide: i.e., $\hat{U} = \hat{U}'$ can be computed from:*

$$\hat{U} = U_i + s^\theta h p_i^\top \nu_{\lambda^*}, \tag{2.57}$$

where $\lambda^* = \operatorname{argmin}_{\lambda \in \mathbb{R}^n} F_0(\lambda)$ and $\nu_{\lambda^*} = p_{\lambda^*} / \|p_{\lambda^*}\|$.

2. *The characteristics found by solving the finite difference problem and minimizing F_0 coincide and are given by $[p_{\lambda^*}, \hat{p}] = [p_{\lambda^*}, 0]$.*

3. *The approximated characteristic passes through* $\mathrm{conv}(\{p_0, \ldots, p_{n-1}\})$ *if and only if* $\lambda^* \in \Delta^n$.

Proof of theorem 4. We assume that U is a linear function in the update simplex; hence, ∇U is constant. By stacking and subtracting eq. 2.54 for different values of i, we obtain, for $i = 0, \ldots, n-1$:

$$\begin{bmatrix} \delta P^\top \\ p_i^\top \end{bmatrix} \nabla U = \begin{bmatrix} \delta U \\ U_0 - \hat{U} \end{bmatrix}. \tag{2.58}$$

The inverse of the matrix in the left-hand side of eq. 2.58 is:

$$\left[\left(I - \frac{\nu_{\min} p_i^\top}{\nu_{\min}^\top p_i} \right) Q R^{-\top}, \quad \frac{\nu_{\min}}{\nu_{\min}^\top p_i} \right], \tag{2.59}$$

which can be checked. This gives us:

$$\nabla U = \left(I - \frac{\nu_{\min} p_i^\top}{\nu_{\min}^\top p_i} \right) Q R^{-\top} \delta U + \frac{U_i - \hat{U}}{\nu_{\min}^\top p_i} \nu_{\min}. \tag{2.60}$$

Hence, $\|\nabla U\|^2$ is a quadratic equation in $\hat{U} - U_i$. Expanding $\|\nabla U\|^2$, a number of cancellations occur since $Q^\top \nu_{\min} = 0$. We have:

$$\delta U^\top R^{-1} Q^\top \left(I - \frac{\nu_{\min} p_i^\top}{\nu_{\min}^\top p_i} \right)^\top \left(I - \frac{\nu_{\min} p_i^\top}{\nu_{\min}^\top p_i} \right) Q R^{-\top} \delta U = \left\| R^{-\top} \delta U \right\|^2 + \frac{\left(p_i^\top Q R^{-\top} \delta U \right)^2}{\|p_{\min}\|^2}, \tag{2.61}$$

so that, written in standard form:

$$\begin{aligned} \left(\hat{U} - U_i \right)^2 + 2 p_i^\top Q R^{-\top} \delta U (\hat{U} - U_i) \; + \; \left(p_i^\top Q R^{-\top} \delta U \right)^2 + \\ \|p_{\min}\|^2 \left(\left\| R^{-\top} \delta U \right\|^2 - \left(s^\theta h \right)^2 \right) = 0. \end{aligned} \tag{2.62}$$

Solving for $\hat{U} - U_i$ gives:

$$\hat{U} = U_i - p_i^\top Q R^{-\top} \delta U + \|p_{\min}\| \sqrt{(s^\theta h) - \|R^{-\top} \delta U\|^2}, \tag{2.63}$$

establishing eq. 2.56.

Next, to show that $\hat{U}' = \hat{U}$, we compute:

$$\begin{aligned}
\hat{U}' &= U_0 + \delta U^\top \lambda^* + s^\theta h \|p_{\lambda^*}\| \\
&= U_0 - \left(Q^\top p_0 + \|p_{\lambda^*}\| R^{-\top} \frac{\delta U}{s^\theta h} \right)^\top R^{-\top} \delta U + s^\theta h \|p_{\lambda^*}\| && \text{(eq. 2.39)} \\
&= U_0 - p_0^\top Q R^{-\top} \delta U + s^\theta h \|p_{\lambda^*}\| \left(1 - \left\| R^{-\top} \frac{\delta U}{s^\theta h} \right\|^2 \right) \\
&= U_0 - p_0^\top Q R^{-\top} \delta U + \|p_{\min}\| \sqrt{(s^\theta h)^2 - \|R^\top \delta U\|^2} = \hat{U}. && \text{(eq. 2.38)}
\end{aligned}$$

To establish eq. 2.57, first note that $-R^{-\top} \delta U = s^\theta h Q^\top \nu_{\lambda^*}$ by optimality. Substituting this into eq. 2.56, we first obtain:

$$\hat{U} = U_i + \frac{s^\theta h}{\|p_{\lambda^*}\|} \left(p_i^\top \operatorname{proj}_{\delta P} p_{\lambda^*} + \|p_{\min}\| \sqrt{p_{\lambda^*}^\top \operatorname{proj}_{\delta P}^\perp p_{\lambda^*}} \right). \tag{2.64}$$

Now, using the notation for weighted norms and inner products, we have:

$$p_i^\top \operatorname{proj}_{\delta P} p_{\lambda^*} + \|p_{\min}\| \sqrt{p_{\lambda^*}^\top \operatorname{proj}_{\delta P}^\perp p_{\lambda^*}} = \langle p_i, p_{\lambda^*} \rangle_{\operatorname{proj}_{\delta P}} + \|p_i\|_{\operatorname{proj}_{\delta P}^\perp} \|p_{\lambda^*}\|_{\operatorname{proj}_{\delta P}^\perp}. \tag{2.65}$$

Since $\operatorname{proj}_{\delta P}^\perp$ orthogonally projects onto $\operatorname{range}(\delta P)^\perp$, and since the dimension of this subspace is 1, $\operatorname{proj}_{\delta P}^\perp p_i$ and $\operatorname{proj}_{\delta P}^\perp p_{\lambda^*}$ are multiples of one another and their directions coincide (see figure 2.6); furthermore, the angle between them is since our simplex is nondegenerate. So, by Cauchy-Schwarz:

$$\|p_i\|_{\operatorname{proj}_{\delta P}^\perp} \|p_{\lambda^*}\|_{\operatorname{proj}_{\delta P}^\perp} = \langle p_i, p_{\lambda^*} \rangle_{\operatorname{proj}_{\delta P}^\perp}. \tag{2.66}$$

Combining eq. 2.66 with eq. 2.65 and cancelling terms yields:

$$p_i^\top \operatorname{proj}_{\delta P} p_{\lambda^*} + \|p_{\min}\| \sqrt{p_{\lambda^*}^\top \operatorname{proj}_{\delta P} p_{\lambda^*}} = p_i^\top p_{\lambda^*}. \tag{2.67}$$

Eq. 2.57 follows.

To parametrize the characteristic found by solving the finite difference problem, first note that the characteristic arriving at $\hat{p}$ is colinear with $\nabla \hat{U}$. If we let $\tilde{\nu}$ be the normal pointing from $\hat{p}$ in the direction of the arriving characteristic, let $\tilde{p}$ be the

point of intersection between $p_0 + \text{range}(\delta P)$ and $\text{span}(\tilde{\nu})$, and let $\tilde{l} = \|\tilde{p}\|$, then, since $\tilde{p} - p_0 \in \text{range}(\delta P)$:

$$\nu_{\min}^{\top}(\tilde{p} - p_0) = 0. \tag{2.68}$$

Rearranging this and substituting $\tilde{p} = \tilde{l}\tilde{\nu}$, we get:

$$\tilde{l} = \frac{\nu_{\min}^{\top} p_0}{\nu_{\min}^{\top} \tilde{\nu}}. \tag{2.69}$$

Now, if we assume that we can write $\tilde{p} = \delta P \tilde{\lambda} + p_0$ for some $\tilde{\lambda}$, then:

$$\tilde{\lambda} = R^{-1} Q^{\top} (\tilde{p} - p_0) = -R^{-1} Q^{\top} \left(I - \frac{\tilde{\nu} \nu_{\min}^{\top}}{\tilde{\nu}^{\top} \nu_{\min}} \right) p_0. \tag{2.70}$$

To see that $\tilde{p} = p_{\lambda^*}$, note that since $\tilde{\nu} = -\nabla \hat{U} / \left\| \nabla \hat{U} \right\| = -\nabla \hat{U} / (s^{\theta} h)$:

$$\text{proj}_{\delta P} \tilde{\nu} = \frac{-\text{proj}_{\delta P} \nabla \hat{U}}{s^{\theta} h} = \frac{-Q R^{-\top} \delta U}{s^{\theta} h} = \text{proj}_{\delta P} \nu_{\lambda^*}. \tag{2.71}$$

Since $\tilde{\nu}$ and ν_{λ^*} each lie in the unit sphere on the same side of the hyperplane spanned by δP, and since $\text{proj}_{\delta P}$ orthogonally projects onto $\text{range}(\delta P)$, we can see that in fact $\tilde{\nu} = \nu_{\lambda^*}$. Hence, $\tilde{p} = p_{\lambda^*} \in p_0 + \text{range}(\delta P)$. The second and third parts of theorem 4 follow. □

2.8 Causality

Dijkstra-like methods are based on the idea of monotone causality, similar to Dijkstra's method itself. To compute shortest paths in a network, Dijkstra's method uses dynamic programming to compute globally optimal shortest paths using local information [44]. In this way, the distance to each downwind vertex must be greater than its upwind neighboring vertices. To ensure convergence to the correct viscosity solution, our scheme must be consistent and monotone [39]. Our OLIMs using the `rhr` quadrature rule inherit the consistency and causality of the finite difference methods which they are equivalent to if they use the same 4 (in 2D) or 6 (in 3D) point neighborhoods. Since we consider many different update neighborhoods involving distinct simplexes, we provide a simple way of checking whether each simplex is causal.

The causality of an update depends on the underlying simplex and the problem data. In particular, an update is causal for F_i if:

$$\hat{U} = F_i(\lambda_i^*) \geq \max_i U_i. \tag{2.72}$$

It is enough to determine whether or not each type of update simplex admits only causal updates, which relates to whether the simplex is acute.

We also consider something we refer to here as the "update gap": the difference $\hat{U} - \max_i U_i$. As discussed in Tsitsiklis's original paper [134], an alternative to Dijkstra's algorithm is Dial's algorithm—a bucketed version of Dijkstra's algorithm which runs in $O(N^n)$ time, where the constant depends on the bucket size [43, 70]. In this case, the size of the buckets is determined by the update gap. It is unclear whether there is any real advantage of a Dial-like solver (see [67] for a discussion), although a new numerical study suggests that there may indeed be some advantage in using a Dial-like solver [70, 58]. Despite this, the update gap is of fundamental importance and limits the number of nodes that can be processed in parallel without violating causality.

Theorem 5. *For $\nu_i = p_i/\|p_i\|$, an update simplex is causal for F_0 if and only if $\nu_i^\top \nu_j \geq 0$ for all i and j such that $0 \leq i < n$ and $0 \leq j < n$. The update gap is given explicitly by:*

$$\hat{U} - \max_i U_i = s^\theta h \min_{i,j} \frac{\nu_i^\top \nu_j}{\|p_i\|}. \tag{2.73}$$

If we assume that s is Lipschitz continuous, then for h small enough, the simplex is also causal for F_1, and the term in F_1 which prevents an update from being causal decays with order $O(h^2)$.

Proof of theorem 5. For causality of F_0, we want $\hat{U} \geq \max_i U_i$, which is equivalent to $\min_i(\hat{U} - U_i) \geq 0$. From eq. 2.56, we have:

$$\min_i \left(\hat{U} - U_i\right) = s^\theta h \min_i \min_{\lambda \in \Delta^n} \frac{\nu_i^\top \nu_\lambda}{\|p_\lambda\|} = s^\theta h \min_{i,j} \frac{\nu_i^\top \nu_j}{\|p_i\|} \geq 0. \tag{2.74}$$

The last equality follows because minimizing the cosine between two unit vectors is equivalent to maximizing the angle between them; since λ is restricted to lie in Δ^n, this clearly happens at a vertex since the minimization problem is a linear program.

For F_1, first rewrite s_λ^θ as follows:

$$s_\lambda^\theta = s^\theta + \theta(s_0 + \delta s^\top \lambda - \overline{s}), \tag{2.75}$$

where $\overline{s} = n^{-1}\sum_{i=0}^{n-1} s_i$. If $\lambda_0^\star$ and $\lambda_1^\star$ are the minimizing arguments for F_0 and F_1, respectively, and if $\delta\lambda^* = \lambda_1^* - \lambda_0^*$, then we have:

$$F_1(\lambda_1^*) = F_0(\lambda_1^*) + \theta\left(s_0 + \delta s^\top \lambda_1^* - \overline{s}\right) h\|p_{\lambda_1^\star}\|. \tag{2.76}$$

By the optimality of λ_0^* and strict convexity of F_0 (lemma 2), we can Taylor expand and write:

$$F_0(\lambda_1^*) = F_0(\lambda_0^*) + \nabla F_0(\lambda_0^*)^\top \delta\lambda^* + \frac{1}{2}\delta\lambda^{*\top}\nabla^2 F_0(\lambda_0^*)\delta\lambda^* + R \geq R, \tag{2.77}$$

where $|R| = O(h^3)$ by theorem 2. Let $\hat{U} = F_1(\lambda_1^*)$. Since F_0 is causal, we can write:

$$\hat{U} \geq \max_i U_i + R + \theta\left(s_0 + \delta s^\top \lambda_1^* - \overline{s}\right) h\|p_{\lambda_1^*}\|. \tag{2.78}$$

Since s is Lipschitz, the last term is $O(h^2)$—in particular, $\|\delta s\| = O(h)$ and $\|s_0 - \overline{s}\| = O(h)$ since s_0 and $\overline{s}$ lie in the same simplex. So, because the gap $\min_i(\hat{U} - U_i)$ is $O(h)$, we can see that $\hat{U} \geq \max_i U_i$ for h sufficiently small. □

In section 2.10 we present a table of update gaps for all update simplices considered in this paper.

The fact that our methods are causal for all practical problem sizes follows from the fact that the term preventing causality decays rapidly—see eq. 2.76. This can be seen easily by rewriting $F_1(\lambda)$ as $F_0(\lambda)$ plus a small perturbation (which is $O(h^2)$) and using the Lipschitz continuity of s.

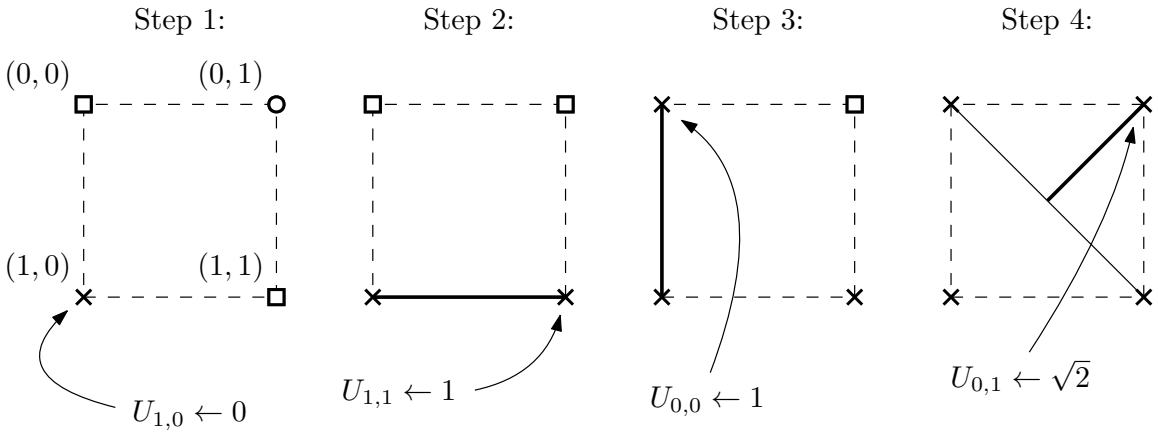


Figure 2.7: *An example of running* `olim4_rhr` *with local factoring for three steps.* Note that `olim4_rhr` is equivalent to the standard 2D fast marching method. We assume $s \equiv 1$. Initially, $p^\circ = p_{1,0}$ is the only node in `bd` and is factored. The first two steps proceed exactly the same as for the unfactored method, since the steps between the source nodes and the updated nodes lie along characteristics. In the third step, the unfactored method would set $U_{0,1} \leftarrow 1 + \sqrt{2}/2$. However, for the factored solver, we find that $U_{0,1} \leftarrow \sqrt{2}/2 + \sqrt{2}/2 = \sqrt{2}$.

2.9 Local factoring

Near rarefaction fans, for example if D is a point source or the domain contains obstacles with corners, the rate of convergence of the eikonal equation is diminished. For the eikonal equation with point source data and constant slowness, this degrades the rate of convergence to $O(h \log h^{-1})$ [106, 143]. Fast sweeping methods for the factored eikonal equations have been developed [52, 81]; likewise, fast marching methods have been developed, and have also been used in the context of travel time tomography [106, 131].

In this section, we show how the ordered line integral method can be easily adapted to additive factoring, and provide numerical tests that show that it recovers the expected linear rate of convergence for factored point source problems. Our focus is locally factored point sources, but this approach can be applied to the globally factored equation and other types of rarefaction fans occuring at corners or discontinuities [106].

Let $x^\circ \in \Omega$ be the location of a point source so that $\texttt{bd} = \{x^\circ\}$, define p° to be the image of x° under the same transformation that takes $\hat{x}$ to $\hat{p}$, and let $s^\circ = s(x^\circ)$. The additive factorization of U around x° is [81, 106]:

$$U(x) = T(x) + \tau(x), \qquad \text{where} \qquad T(x) = s^\circ \left\| x - x^\circ \right\|, \tag{2.79}$$

i.e. $u_\lambda = T_\lambda + \tau_\lambda$ where $T_\lambda = s^\circ h \| p_\lambda - p^\circ \|$. Our original definition of F_i was such that $\hat{U} = F_i(\lambda^*)$. We will define G_i analogously. Letting $\tau_\lambda = \tau_0 + \delta\tau^\top \lambda$, where τ_i and T_i are the values of τ and T at p_i for each i, we define:

$$G_0(\lambda) = \tau_\lambda + T_\lambda + s^\theta h \|p_\lambda\|, \tag{2.80}$$

$$G_1(\lambda) = \tau_\lambda + T_\lambda + s_\lambda^\theta h \|p_\lambda\|. \tag{2.81}$$

Like with F_0 and F_1, the only difference between G_0 and G_1 is between the terms containing s^θ and s_λ^θ. We do not explicitly refer to them in the rest of this paper, but the cost functions $G_{\texttt{rhr}}$, $G_{\texttt{mp0}}$, and $G_{\texttt{mp1}}$ are defined analogously to F_{rhr}, F_{mp0}, and F_{mp1} from the θ-rules G_0 and G_1.

To solve the factored eikonal equation, we choose a factoring radius r°, replacing F_i with G_i in eq. 2.12 for nodes which lie within a distance r° of x°. For constant slowness, the effect of this is to solve eq. 1.4 exactly inside of the locally factored region. For clarity, this is depicted in figure 2.7. Algorithm 2 can be applied to solve eq. 2.12 for factored nodes. The gradient and Hessian of G_i are simple modifications of the gradient and Hessian for F_i.

Lemma 6. *The gradient and Hessian of G_i for $i = 0, 1$ are given by:*

$$\nabla G_i(\lambda) = \nabla F_i(\lambda) - \delta\tau + \frac{s^\circ h}{\|p_\lambda - p^\circ\|} \delta P^\top (p_\lambda - p^\circ), \tag{2.82}$$

$$\nabla^2 G_i(\lambda) = \nabla^2 F_i(\lambda) + \frac{s^\circ h}{\|p_\lambda - p^\circ\|} \delta P^\top \operatorname{proj}^\perp_{p_\lambda - p^\circ} \delta P. \tag{2.83}$$

2.10 Implementation of the ordered line integral method

In this section, we describe our *bottom-up* and *top-down* algorithms (see fig. 2.1). We focus on 3D solvers, since in 2D the distinction between the two is less important. Each algorithm reduces the number of updates that are done without degrading solution accuracy by using an efficient enumeration or search for update simplexes. The primary difference between the two algorithms is the ordering of the updates' dimensions: *top-down* proceeds from $d = n - 1$ down to $d = 0$, skipping lower-dimensional updates when possible, and *bottom-up* proceeds from $d = 0$ up to $d = n - 1$, skipping higher-dimensional updates when it can.

Simplex enumeration for the *top-down* algorithm

When a node is first removed from `front` and has just become `valid` (item 3a in algorithm 1), an isotropic solver must do updates involving, at the very least, the node's $2n$ nearest neighbors. We can use larger neighborhoods to improve the accuracy of the result. Doing so does not necessarily improve the order of convergence of the solver, but can significantly improve the error constant of the solution. For all of the solvers considered in this paper, in 3D, we only consider neighborhoods with at most 26 neighbors. See fig. 3.2 for neighborhoods for the *top-down* solver.

For the *top-down* solver, we simplify things by only doing updates which have $p_i = p_{\text{new}}$ for some i. To iterate over all update simplexes, by symmetry, we enumerate all simplexes in a single octant. This means enumerating, e.g., $\binom{7}{3} = 35$ tetrahedra. Some choices lead to degenerate tetrahedra, so the number of nondegenerate update tetrahedra is fewer than 35 per octant. This makes it reasonable to write out the update procedure as straight-line code.

We enumerate the tetrahedra in a type of "shift-order" (see, e.g., [6])—that is, we

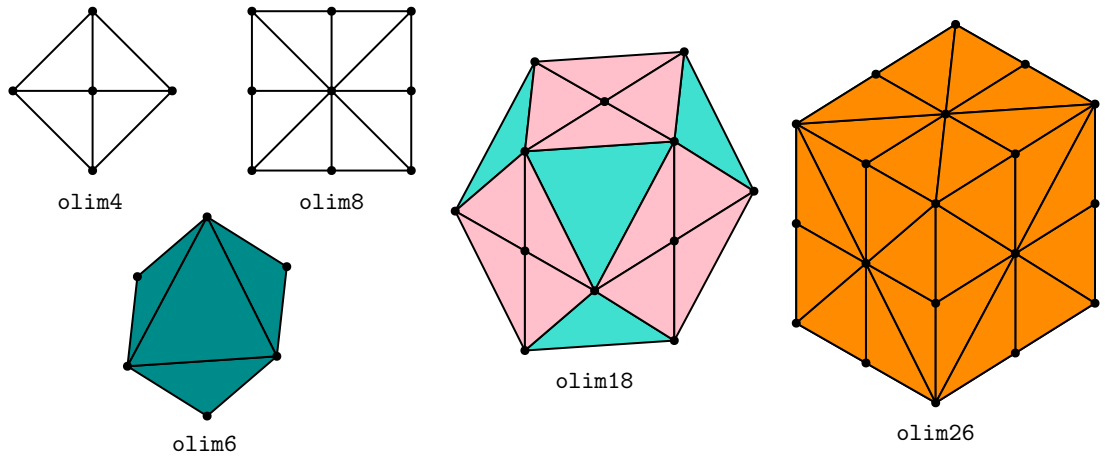


Figure 2.8: *Neighborhoods for the* top-down *family of algorithms.* Algorithms `olim4` and `olim8` are 2D solvers and the rest are 3D solvers. The color coding of tetrahedron updates is the same for this figure and figure 2.9 below.

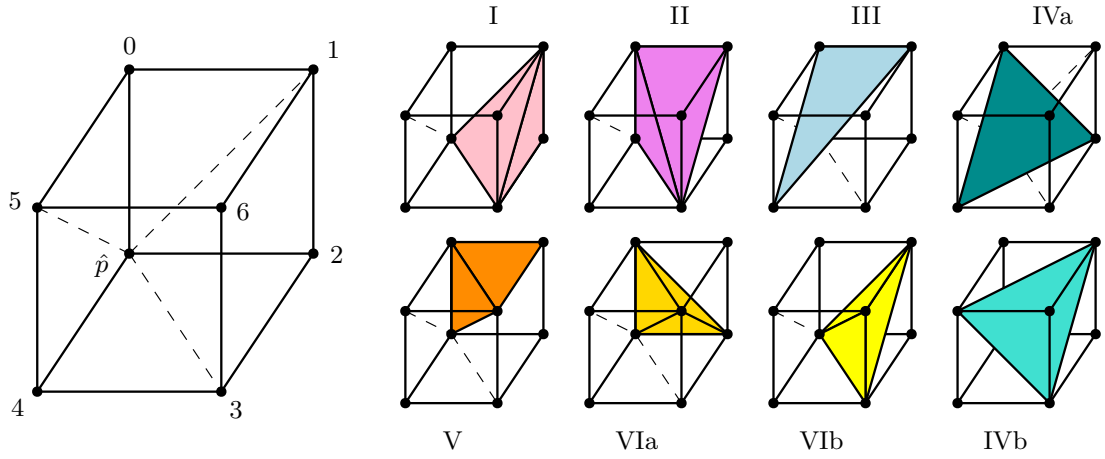


Figure 2.9: *Numbering scheme for update groups for the* top-down *solvers.* In this diagram, $\hat{p}$ is being updated. The diagonally opposite node is the sixth (last) node, with the other six nodes numbered 0–5 cyclically.

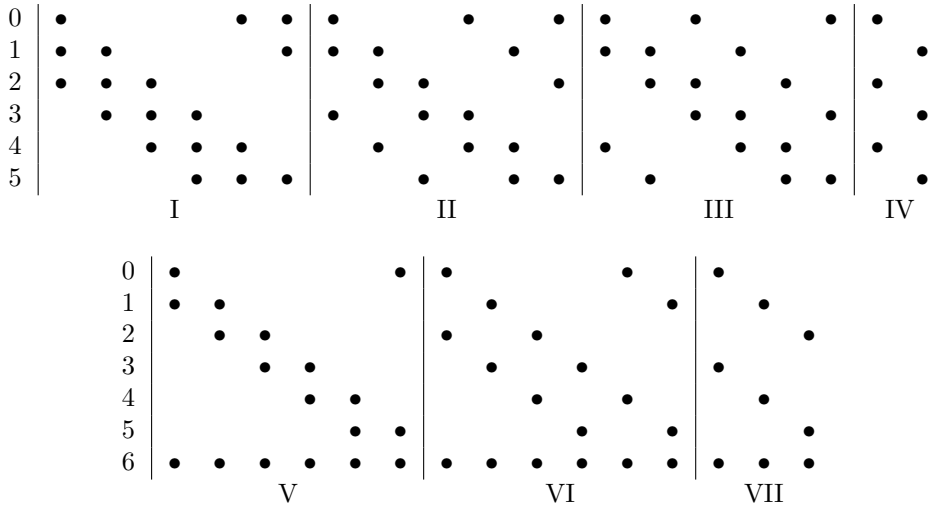


Figure 2.10: *Tables of update groups.* These tables should be scanned columnwise: each column of dots selects a different tetrahedron. Tetrahedra (0, 1, 2), (2, 3, 4), and (4, 5, 0) in group I and all tetrahedra in group VII are degenerate and can be omitted.

start with an unseen bit pattern, and group this pattern together with all of its shifts (with rotation). This groups the tetrahedra into sets that are rotationally symmetric about the diagonal of the octant. In our implementation, we conditionally compile different groups so that no unnecessary branching is done. This is done using C++ templates [128]. Example stencils for the versions of `olim6`, `olim18`, and `olim26` that are used for our numerical test are shown in figure 3.2. The tetrahedron groups are shown in figures 2.9 and 2.10.

Update gaps for tetrahedron groups

If we apply theorem 5 to the tetrahedron groups enumerated in figures 2.9 and 2.10, we get the following update gaps, enumerated in the same order as fig. 2.9 (ignoring the $s^\theta h$ factor):

Group I	$1/\sqrt{2}$	Group II	$1/\sqrt{2}$	Group III	$1/\sqrt{2}$	Group IVa	0
Group V	$1/\sqrt{3}$	Group VIa	0	Group VIb	$\mathbf{2/\sqrt{3}}$	Group IVb	$1/\sqrt{2}$

The update gap is first explored in Tsitsiklis's original paper [134]; in this work, the fact that Group IVa has no update gap and that the update gap of Group V is $1/\sqrt{3}$ is noted and an $O(N^n)$ algorithm based on Dial's algorithm is presented using Group V for the update tetrahedra. This same observation is made in a more recent paper about a method based on Dial's algorithm [70]. A method based on a combination of tetrahedra groups will have an update gap that is the minimum of each of the individual groups' gaps. We note here that a solver based on a combination of Groups I and VIb has a larger update gap than a solver based on Group V. This should have a positive impact on the performance of any parallel Dijkstra-like method.

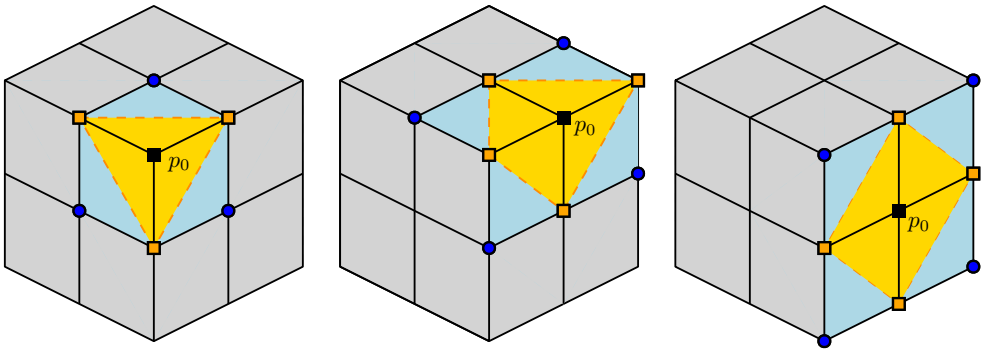


Figure 2.11: *The three types of neighborhoods for the* bottom-up *algorithm.* The yellow and blue regions indicate where triangle and tetrahedron updates may be performed, respectively. For instance, with p_0 the minimizing line update vertex, candiates for p_1 consist of the yellow nodes: triangle updates involving these candidates and p_0 will be performed. Once a yellow node (p_1) has been selected, tetrahedron updates involving the neighboring blue nodes (candidates for p_2) will be performed. Note that the updates performed correspond roughly to a combination of groups I, V, VIa, and VIb.

The search procedure used by the *bottom-up* algorithm

Another approach is to use local characteristic information obtained by performing lower-dimensional updates to help us avoid performing unnecessary higher-dimensional updates. Intuitively, if we find the minimum line update ($d = 0$), then we can avoid triangle updates ($d = 1$) that don't include the minimizing line update. Since we only perform updates for $d > 0$ that include the newly `valid` node p_{new}, we can start our search with $d = 1$.

After doing an update of dimension d, we find neighboring updates of dimension $d+1$. For $n = 3$ and $d = 1$, for each p_0, we find points p_1 that satisfy $\|p_0 - p_1\|_1 \leq 1$. For $d = 2$, for each pair (p_0, p_1), we find points p_2 that satisfy $\|p_0 - p_2\|_1 \leq 2$ and $\|p_1 - p_2\|_1 \leq 2$ simultaneously. In practice, we only find these neighbors once and precompute an array of indices to be used later. The neighborhoods computed in this way are shown in fig. 2.11.

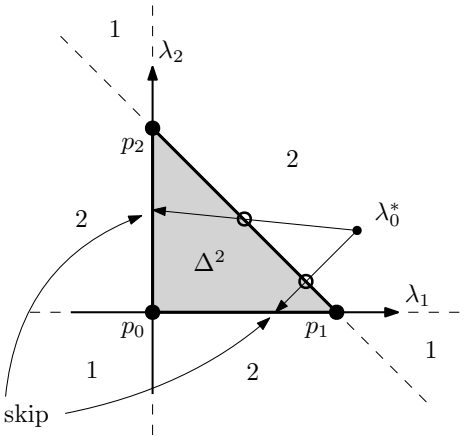


Figure 2.12: *Skipping lower-dimensional updates when solving the unconstrained minimization problem.* For $d = 2$, if $\lambda_0^* \in \Delta^2$, all three triangle updates can be skipped. On the other hand, when minimizing F_0 using theorem 3, if $\lambda_0^* \notin \Delta^2$ and depending on where λ_0^* lies, it is possible to skip one or two triangle updates. In this case, we label the different regions by the number of updates that it is possible to skip: λ^* here lies in "zone 2", since it is possible to skip the two triangle updates on the opposite side of Δ^2. Along the same lines, if λ^* were to lie in "zone 1", two triangle updates would be "visible", and it would only be possible to skip one update.

Minimization algorithms and skipping updates

When $F_i = F_0$, we can use theorem 3 or 4 to compute λ_0^*. If $F_i = F_1$, we need to use an algorithm that can solve the constrained optimization problem defined by eq. 2.12. Our approach has been to use sequential quadratic programming (SQP), although there are many other options [17, 94]. It is possible to skip some updates; and, indeed, the performance of our algorithms comes about largely because of this happening.

We skip updates in three different ways. The first two approaches are used by the *top-down* algorithms, and the third by the *bottom-up* algorithms.

Top-down constrained skipping. When computing an update using a constrained solver, we can rule out all incident lower-dimensional updates, since we have computed the global constrained optimum on Δ^d.

Top-down unconstrained skipping. If we do an update using an unconstrained solver, then depending on where the optimum λ_0^* lies, we can skip some or all lower-dimensional updates. The idea is simple and is best depicted visually, see fig. 2.12.

Bottom-up KKT skipping. We can also skip higher-dimensional updates. For example, if we do the three triangle updates on the boundary of a tetrahedron update, we can use the Karush-Kuhn-Tucker necessary conditions for optimality of a constrained optimization problem [94] to determine if the minimizer on the boundary is also a global minimizer for the constrained minimization problem given by eq. 2.12. See Section 2.11. We note that a modified version of this strategy for skipping updates was used in the work on computing the quasipotential for nongradient SDEs in 3D [140].

2.11 Skipping updates in the *bottom-up* family of algorithms

In this section, we describe how to use the KKT conditions to skip updates in the *bottom-up* algorithms. In this section, we write:

$$A = \begin{bmatrix} -1 & & \\ & \ddots & \\ & & -1 \\ 1 & \cdots & 1 \end{bmatrix} \in \mathbb{R}^{d+1 \times d}, \qquad b = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \in \mathbb{R}^{d+1} \tag{2.84}$$

Using these, the set Δ^d can be written as a linear matrix inequality:

$$\lambda \in \Delta^d \iff A\lambda \leq b \tag{2.85}$$

Let $\mu \in \mathbb{R}^{d+1}$ be the vector of Lagrange multipliers. Then, the Lagrangian function for eq. 2.12 is:

$$L(\lambda, \mu) = F(\lambda) + (A\lambda - b)^\top \mu. \tag{2.86}$$

Since F_0 is strictly convex and since we assume h is small enough for F_1 to be strictly convex, if λ^* lies on the boundary of Δ^d, we only need to check that the optimum Lagrange multipliers μ^* are dual feasible; i.e., whether $\mu^* \geq 0$ (this follows directly from the standard KKT conditions [17, 94]). For a fixed $\lambda \in \Delta^d$, define the set of indices of active constraints:

$$\mathcal{I} = \{i : (A\lambda - b)_i = 0\} \tag{2.87}$$

That is, $i \in \mathcal{I}$ if the ith inequality holds with equality ("is active"). Stationarity then requires:

$$A_{\mathcal{I}}^{\top} \mu_{\mathcal{I}}^* = \nabla F_i(\lambda). \tag{2.88}$$

If $i \notin \mathcal{I}$, we set $\mu_i^* = 0$. If $\mu_i^* \geq 0$ for all i, then the update may be skipped.

When implementing this, since A is sparse, it is simplest and most efficient to write out the system given by eq. 2.88 and write a specialized function to solve it. Note that since we always start with a lower-dimensional interior point solution lying on the boundary of a higher-dimensional problem, we only have to compute one Lagrange multiplier.

The *bottom-up* and *top-down* algorithms

In this section, we describe our *bottom-up* and *top-down* algorithms. We note for clarity that these algorithms correspond to the "compute $\hat{U}$" part of item 3d of alg. 1.

To describe our *top-down* algorithm (see algorithm 2), we define the set of admissible d-dimensional updates neighboring the point $\hat{p}$ by:

$$\begin{aligned} \mathcal{V}_d = \big\{ \{p_0, \ldots, p_d\} : p_i\texttt{.state} = \texttt{valid} \text{ for } i = 0, \ldots, d, \\ \text{and } \{p_0, \ldots, p_d\} \text{ are in a selected update group,} \\ \text{and } p_{\text{new}} \in \{p_0, \ldots, p_d\} \big\} \end{aligned} \tag{2.89}$$

Algorithm 2 The *top-down* algorithm.

1. Set $\hat{U} \leftarrow \infty$.
2. Initialize $\mathcal{V}_d$ according to eq. 2.89 for each $d = 0, \ldots, n-1$.
3. For $d = n-1$ down to 0:
   a) For each $(p_0, \ldots, p_d) \in \mathcal{V}_d$:
      i. If $F_i = F_0$ (`mp0` or `rhr`):
         A. Compute U for $(p_0, \ldots, p_d)$ using theorem 3 or 4.
         B. Remove updates from $\mathcal{V}_0, \ldots, \mathcal{V}_{d-1}$ by visibility (see figure 2.12).
      ii. Otherwise, if $F_i = F_1$ (`mp1`):
         A. Compute U by solving eq. 2.12 numerically (we use SQP).
         B. Remove all lower-dimensional updates from $\mathcal{V}_0, \ldots, \mathcal{V}_{d-1}$.
      iii. Set $\hat{U} \leftarrow \min(\hat{U}, U)$.

Algorithm 3 The *bottom-up* algorithm.

1. Set $\hat{U} \leftarrow \infty$ and $p_0 \leftarrow p_{\text{new}}$.
2. For $i = 1, \ldots, n-1$:
   a) For each `valid` p_i close enough to $p_0, \ldots, p_{i-1}$ (see section 2.10), do the update corresponding to $(p_0, \ldots, p_i)$ and keep track of the minimizing $\lambda^* \in \Delta^i$. This update can optionally be skipped by first computing μ^* corresponding to the optimum of the incident lower-dimensional update $(p_0, \ldots, p_{i-1})$ and checking if $\mu^* \geq 0$.
   b) Let p_i be the node which forms the update with the minimum value.
   c) If $F_i = F_0$ (`mp0` or `rhr`), compute U for $(p_0, \ldots, p_i)$ using theorem 3 or 4.
   d) Otherwise, if $F_i = F_1$ (`mp1`), compute U for $(p_0, \ldots, p_i)$ by solving eq. 2.12.
   e) Set $\hat{U} \leftarrow \min(\hat{U}, U)$.

for $d = 0, \ldots, n-1$. The update set $\mathcal{V}_d$ collects all admissible simplex updates of a given dimension: i.e., updates which both belong to a group as defined in section 2.10 and are `valid`. The third condition is an important optimization. To see why it is correct, fix an update set $\mathcal{V}_d$. If $\{p_0, \ldots, p_d\}$ satisfies the first two conditions but not the third, we can see that $\hat{p}$ would have already been updated from it in a previous iteration. All new information affecting $\hat{U}$ during this iteration must be computed from an update involving p_{new}.

The *bottom-up* algorithm (algorithm 3) builds up each update $(p_0, \ldots, p_d)$ one

vector at a time by searching for adjacent minimizing updates of higher dimension. The optimization involving p_{new} described above can be incorporated by initially setting $p_0 \leftarrow p_{\text{new}}$.

2.12 Numerical Results

We do tests involving several different slowness functions with analytic solutions for point source data, and a linear speed function (i.e., $1/s$) which has been shown to be amenable to local factoring. For each quadrature rule described in section 2.3 (`mp0`, `mp1`, or `rhr`), we have two 2D algorithms, `olim4` and `olim8`, corresponding to 4- and 8-point stencils, respectively. Since there is no advantage in 2D, we don't apply the *top-down* or *bottom-up* approaches. In 3D, we have three *top-down* algorithms: `olim6` (group IVa), `olim18` (groups I, IVa, and IVb), and `olim26` (group V). We also test the *bottom-up* algorithm `olim3d` (see figure 2.11).

Implementation Notes

Before describing our numerical tests, we briefly comment on our implementation and make some observations about its performance. A discussion of some of the choices that we made in our implementation follows:

- We precompute and cache all values of s on the grid $\mathcal{G}$, as opposed to reevaluating s, because we assume that s will be provided as gridded data (consider, e.g., the shape from shading problem [73], where the input data is an image).
- We maintain `front` using a priority queue implemented using an array-based binary heap, which is updated using the `sink` and `swim` functions described in Sedgewick and Wayne [114].

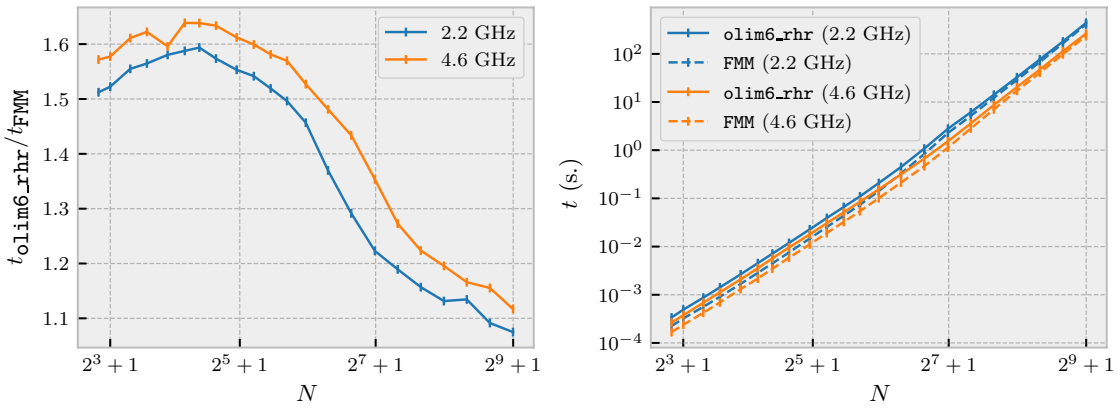


Figure 2.13: *Slowdown incurred by using* `olim6_rhr` *instead of the FMM.* From left to right: 1) the ratio of runtimes versus N, 2) the total CPU runtime of each solver. We compare results on two different computers: "2.2 GHz" is a 2015 MacBook Air with a 2.2 GHz Intel Core i7 CPU, 8 GB of 1600 MHz DDR3 RAM, a 256 KiB L2 cache, and a 4 MiB L3 cache; "4.6 GHz" is a custom built workstation running Linux with a 4.6 GHz Intel Core i7 CPU, 64 GB of 2133 MHz DDR4 RAM, a 1536 KiB L2 cache, and 12 MiB L3 cache. Both computers have 32 KiB L1 instruction caches and data caches. The plots here use our standard $\Omega = [-1, 1]^3$ domain discretized into $N = 2^p + 1$ nodes in each direction, with $s \equiv 1$ and a point source at the origin.

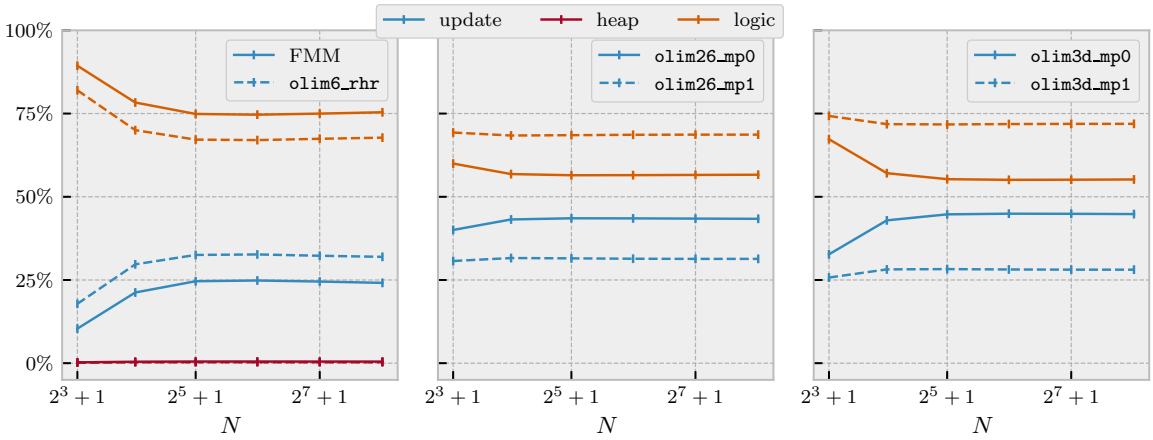


Figure 2.14: *Percentage of time spent on different tasks as determined by profiling.* The "update" tasks and "heap" tasks are clearly defined, while the "logic" task contains a variety of things related to control-flow, finding neighbors, and memory movement—basically, the parts of algorithm 1 that don't clearly pertain to computing new $\hat{U}$ values or keeping `front` updated. From these plots, it is clear that memory speed plays a large role in determining efficiency. To some extent, even though the more complicated update procedures are slower, their slowness is hidden somewhat by memory latency as problem sizes grow. For large N and some solvers (the middle and right plots), "heap" takes too little time, and is not picked up by the profiler.

- We store `front` as a dense grid of states: for each node in $p \in \mathcal{G}$, we track p.`state` for all time for every node. We could implement a sparse `front` using a hash map or a quadtree or octree, which would save space, but would also be much slower to update.

We use a policy-based design written in C++ [4]. This allows us to conditionally compile different features and reuse logic to implement different Dijkstra-like algorithms. In particular, we implement the standard FMM [118] and make a direct comparison between it and the ordered line integral method which it is equivalent to, `olim6_rhr` (see figure 2.13). We have found that only a modest slowdown is incurred by using `olim6_rhr` for problems of moderate size. The disparity between the two is greater for smaller problem sizes, which is due to cache effects. In general, the difference in speed is due to the fact that the FMM's update is extremely simple since it requires only the solution of quadratic equations.

Using Valgrind [92], we profiled running our solver on the numerical tests below for different problem sizes and categorized the resulting profile data. See figure 2.14. The "update" task corresponds to time spent actually computing updates, the "logic" task is a grab bag category for time spent on program logic, and "heap" corresponds to updating the array-based heap which implements `front`. Since the asymptotic complexity of the "update" and "logic" sections is $O(N^n)$, and since "heap" is $O(N^n \log N)$, we can see from figure 2.14 that since so little time is spent updating the heap, the algorithm's runtime is better thought of as $O(N^n)$ for practical problem sizes. This is a consequence of using an array-based heap whose updates are cheap and cache friendly, and a dense grid of states, which can be read from and written to in $O(1)$ time.

As an aside, we mention that thorough numerical studies of eikonal solvers in the literature have been scarce, but we can point out a recent study which seeks to close this gap [58]. Our studies profiling our implementation using Valgrind are carried out

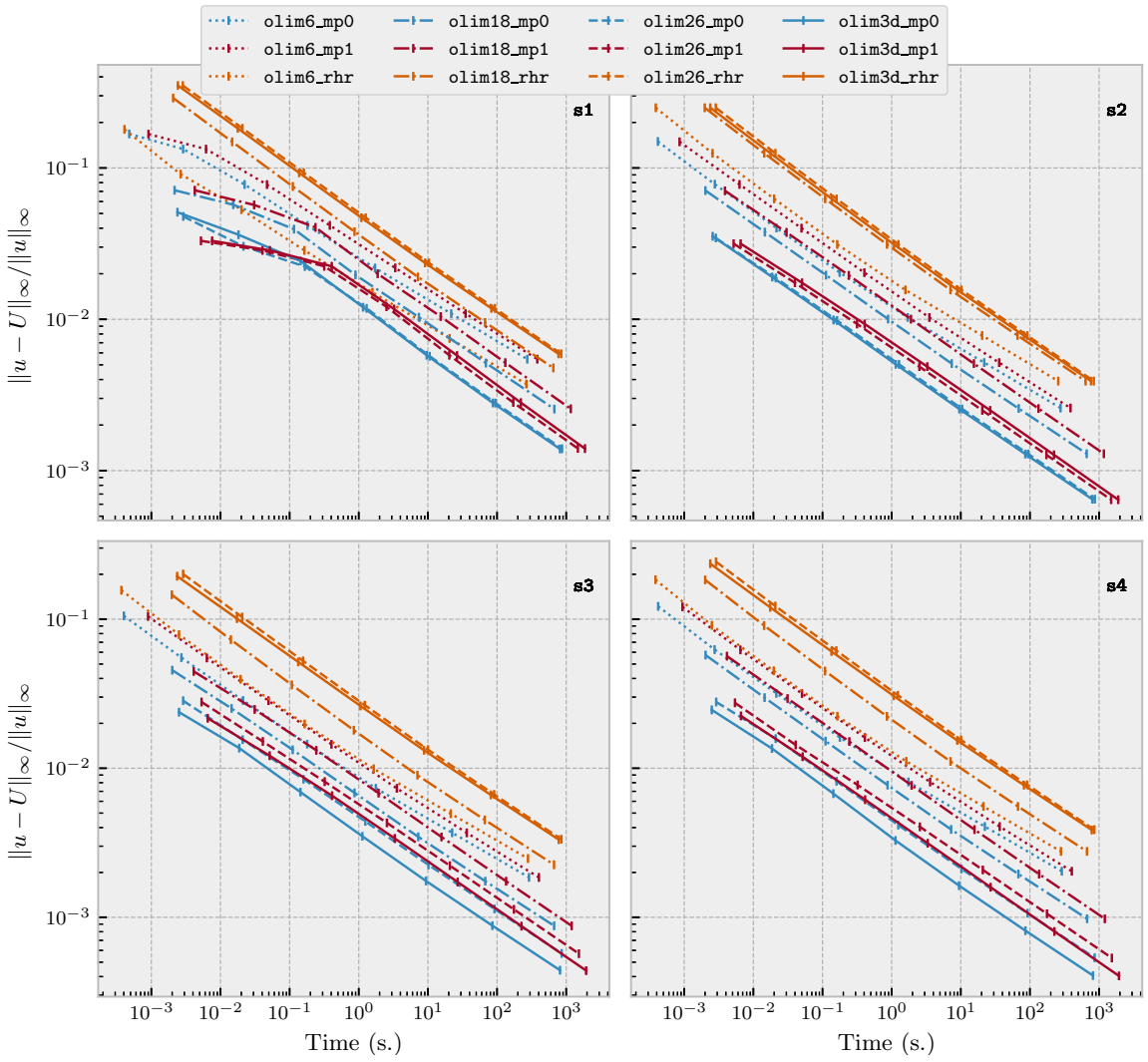


Figure 2.15: *Relative ℓ_∞ error plotted against CPU runtime in seconds.* The domain is $\Omega = [-1, 1]^3$ discretized uniformly in each direction into $N = 2^p + 1$ points, where $p = 3, \ldots, 9$, so that there are N^3 points overall. The slowness functions used are listed in section 2.12. We note that the horizontal and vertical axes of each subplot are the same.

in the same spirit as this work.

Slowness functions with an analytic solution for a point source

Using eq. 1.4 directly, a simple recipe to create pairs of slowness functions and solutions is to prescribe a continuous function u whose level sets are each homeomorphic

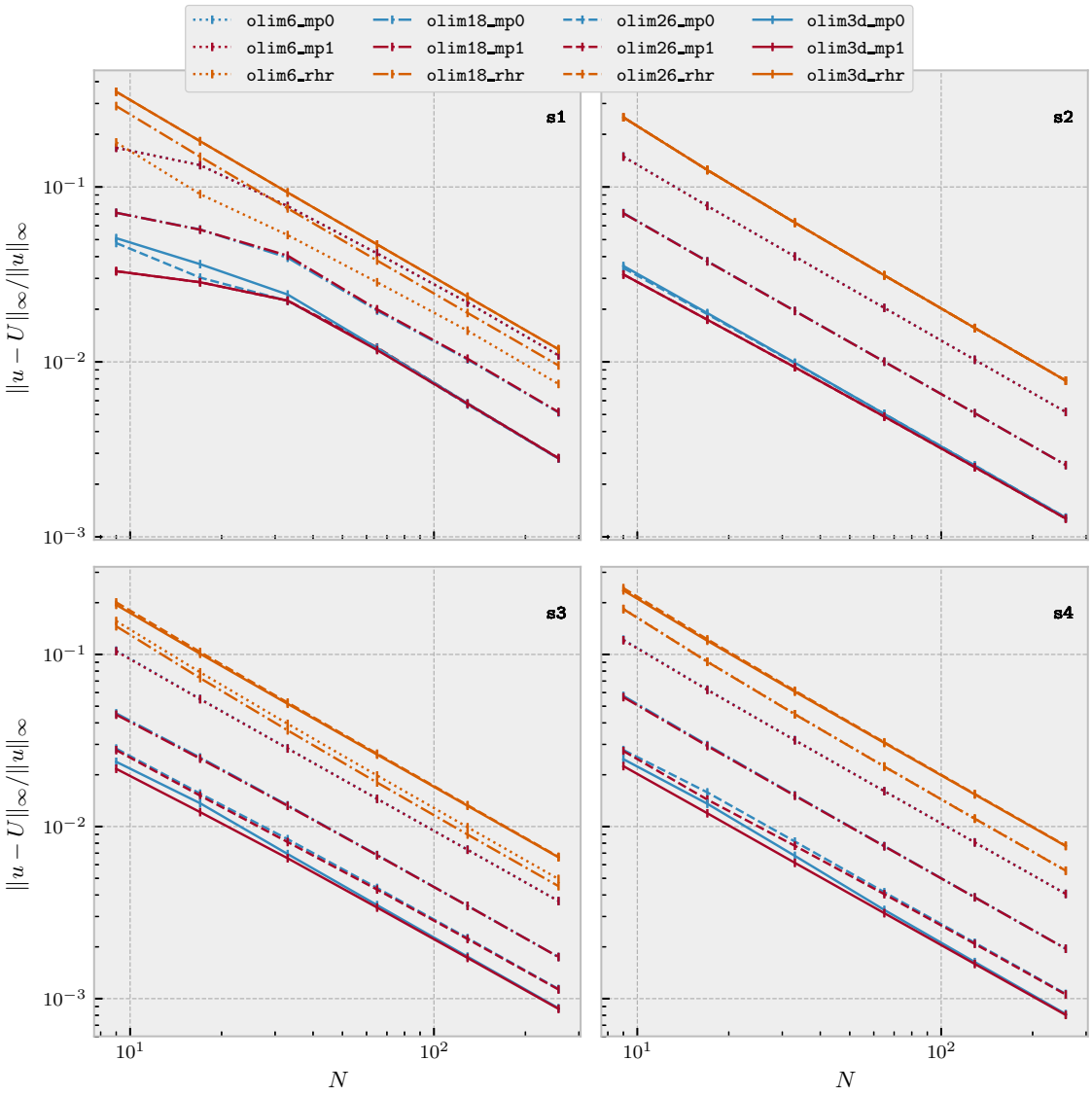


Figure 2.16: *Relative ℓ_∞ error plotted versus N.* The setup is the same as in figure 3.7, except that $p = 3, \ldots, 8$, so that the largest N is 257 instead of 513. For `olim26` and `olim3d`, we can see that `mp0` is initially less accurate than `mp1` but quickly attains parity, in accordance with theorem 2. For `olim6` and `olim18`, the error is the same between `mp0` and `mp1` for all slowness functions, so these plots overlap.

to a ball and compute $s(x) = \|\nabla u(x)\|_2$ analytically, which is valid for a single point source at the origin. Such tests allow us to observe the effect of local factoring, and to see how `mp0`, `mp1`, and `rhr` compare. The following table lists our test functions:

Name	$u(x)$	$s(x)$
`s1`	$\cos(r) + r - 1$	$1 - \sin(r)$
`s2`	$r^2/2$	r
`s3`	$S(x)^\top A S(x)$	$\alpha \left\|\mathrm{diag}(C(x))(A + A^\top)S(x)\right\|$
`s4`	$\frac{1}{2}x^\top A^{1/2} x$	$\|x\|_A = \sqrt{x^\top A x}$

We assume that $x \in \Omega = [-1, 1]^3$. We also define $r = \|x\|$, and vector fields $S(x) = (\sin(\alpha x_i))_{i=1}^3$ and $C(x) = (\cos(\alpha x_i))_{i=1}^3$; we take $\alpha = \pi/5$. For `s3` and `s4`, we assume that A is symmetric positive definite. In 3D, the matrices we use for `s3` and `s4` are:

$$A_{\texttt{s3}} = \begin{bmatrix} 1 & 1/4 & 1/8 \\ 1/4 & 1 & 1/4 \\ 1/8 & 1/4 & 1 \end{bmatrix} = A_{\texttt{s4}}^{1/2} \tag{2.90}$$

Our results are displayed in figures 3.7 and 3.6. We include the relative ℓ_∞ error versus problem size and time, as well as the ℓ_∞ error versus N. We summarize our observations:

- Using either of the midpoint rules (`mp0` or `mp1`) allows improved directional coverage to translate into an improved error constant. See figs. 3.6 and 3.7.

- For `rhr`, increased directional coverage (`olim6_rhr` → `olim18_rhr` → `olim26_rhr`) does not lead to an improved error. In fact, for `s1`, `s3`, and `s4`, increasing the directional coverages causes the accuracy to deteriorate (see figure 3.6). This may be due to the fact that quadrature error and linear interpolation error have different signs, and may partially compensate each other (e.g., in `olim6`). This effect may get worse with increased directional coverage.

- If we scan each graph horizontally, focusing on the plot markers, we can see that the difference in error between `mp0` and `mp1` is minimal. For each `mp1` graph, the corresponding `mp0` graph has the same error, but is shifted to the left, reflecting the fact that the `mp0` OLIMs are substantially faster. This is consistent with theorem 2, which justifies the use of `mp0`. See fig. 3.7.

- With respect to the choice of neighborhood, `olim6` is the fastest; and, for each choice of neighborhood, `mp0` provides the best combination of speed and accuracy. See fig. 3.7. If we are willing to pay somewhat in speed, we can dramatically improve the error constant by improving the directional coverage and using a solver like `olim3d_mp0`. This tradeoff is more pronounced for smaller problem sizes. A theme running through this work is that, as the problem size increases, memory access patterns come to dominate the runtime, and the disparity in runtimes between the faster and slower neighborhoods becomes less pronounced. To see this, compare the start of each graph in the top-left of the plots, and their ends in the bottom-right (again, see fig. 3.7). We can observe, e.g., that the maximum horizontal distance between starting points and ending points has decreased significantly, which confirms this observation.

- Our high accuracy algorithms allow us to obtain a better solution on rough grids: this is helpful since opportunities to refine the mesh are limited in 3D. Discretizing $\Omega = [-1, 1]^2$ in each direction into $N = 2^{14} + 1$ nodes requires about as much memory as discretizing $\Omega = [-1, 1]^3$ with $N = 2^9 + 1$, which leads to h being 32 times smaller in 2D than in 3D.

- In general, `olim26` and `olim3d` are significantly more accurate than `olim6` and `olim18`.

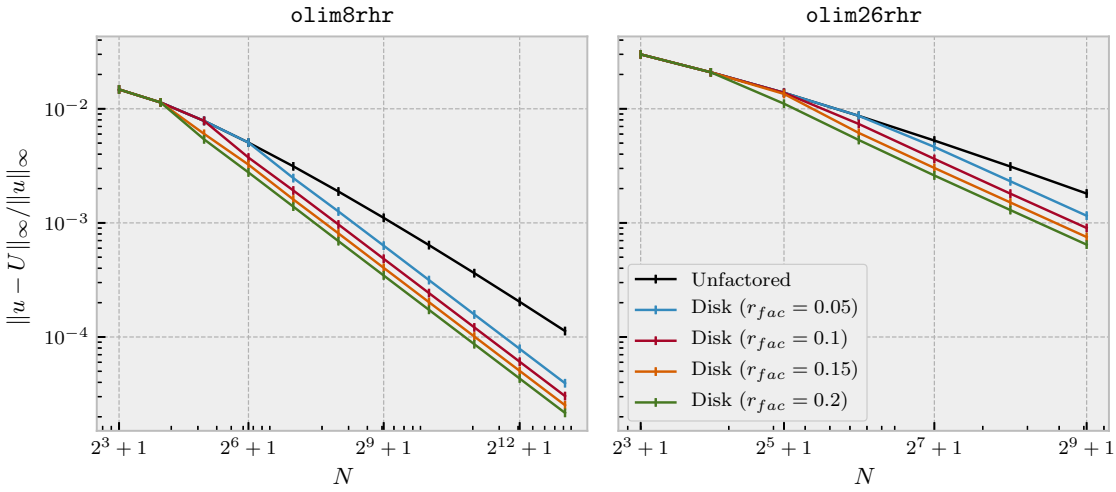


Figure 2.17: *Comparing different ways of selecting factored nodes.* For the test problem, $\Omega = [-1, 1]^n$, with $n = 2$ (left) and $n = 3$ (right). The domain is descretized into N^3 nodes, where $N = 2^p + 1$, so that $h = 2/(N-1)$. The slowness function is constant ($s \equiv 1$). For the 2D problem, `olim8_rhr` is used; `olim26_rhr` is used for the 3D problem. Solutions for the unfactored problem are plotted, along with solutions using a disk/sphere neighborhood with constant factoring radius given by $r^\circ = 0.05, 0.1, 0.15, 0.2$. We note that for this problem the choice $r^\circ = \sqrt{n}$ results in an exact solution. This only applies to the constant slowness function, $s \equiv 1$.

A linear speed function

We consider a problem that has a known analytical solution and has been used as a test problem for other factored eikonal equation solvers before[1] [125, 52, 106]. For a single point source at x_i and a vector v, we define:

$$\frac{1}{s(x)} = \frac{1}{s(x_i)} + v^\top (x - x_i), , \tag{2.91}$$

where $s_i = s(x_i)$. The analytic solution to eq. 1.4 for a single source and slowness function given by eq. 2.91 is [125]:

$$u_i(x) = \frac{1}{\|v\|} \cosh^{-1}\left(1 + \frac{s_i}{2} s(x) \|v\|^2 \|x - x_i\|^2\right). \tag{2.92}$$

If we shift the point source from x_i to another location x_j, we find:

$$\frac{1}{s_i} + v^\top (x - x_j + x_j - x_i) = \frac{1}{s_i} + v^\top (x_j - x_i) + v^\top (x - x_j) = \frac{1}{s_j} + v^\top (x - x_j). \tag{2.93}$$

[1] We thank D. Qi for helpful discussions regarding this problem.

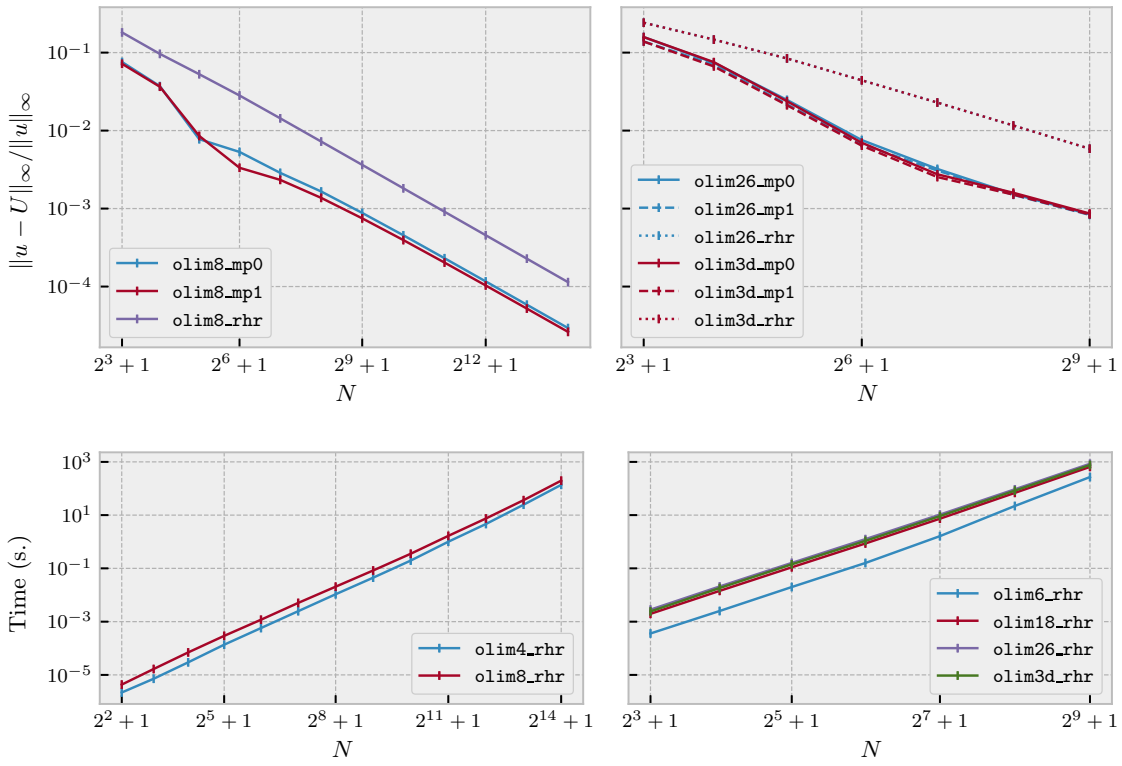


Figure 2.18: *Numerical results for the linear speed function of section 2.12.* Problem sizes are $N = 2^p + 1$, where $p = 3, \ldots, 14$ in 2D and $p = 3, \ldots, 9$ in 3D. The total number of nodes is N^n, where $n = 2, 3$. See section 2.12 for least squares fits. Top row: relative ℓ_∞ error plotted versus N in 2D (left) and 3D (right). Bottom row: wall clock time plotted versus N in 2D (left) and 3D (right).

That is, the slowness function s remains unchanged as it is rewritten with respect to a different source.

If $\{x_i\}$ is a set of point sources and $u_i(x)$ is the solution of the eikonal equation for the single point source problem with point source given by x_i, then the solution for the multiple point source problem with sources $\{x_i\}$ is:

$$u(x) = \min_i u_i(x). \tag{2.94}$$

We use this formula to compare relative ℓ_∞ errors for each of our OLIMs in 2D and `olim26` and `olim3d` in 3D for this slowness function with a pair of point sources, $x_1 = (0, 0)$ and $x_2 = (0.8, 0)$ in 2D, and $x_1 = (0, 0, 0)$ and $x_2 = (0.8, 0, 0)$ in 3D. We set the domain of the problem to be $\Omega = [0, 1]^n$ and discretize it into $N = 2^p + 1$

Neighborhood	C_T	α
`olim4`	7.779×10^{-8}	1.0785
`olim8`	1.971×10^{-7}	1.0515
`olim6`	2.968×10^{-7}	1.085
`olim18`	2.984×10^{-6}	1.018
`olim26`	4.649×10^{-6}	1.0103
`olim3d`	3.923×10^{-6}	1.013

(a) $T_N \sim C_T N^{\alpha n}$

Neighborhood	C_E	β
`olim8_mp0`	0.4077	0.98744
`olim8_mp1`	0.3683	0.993
`olim8_rhr`	1.511	0.9728
`olim26_mp0`	2.328	1.3135
`olim26_mp1`	1.949	1.2888
`olim26_rhr`	1.772	0.90394
`olim3d_mp0`	2.268	1.3141
`olim3d_mp1`	1.865	1.2885
`olim3d_rhr`	1.77	0.90353

(b) $E_N \sim C_E h^{\beta}$

Table 2.1: *Least-squares fits of the runtime and relative ℓ_∞ error for OLIMs in 2D and 3D.* We denote the time for a given N by T_N; likewise, E_N denotes the relative ℓ_∞ error for a specific N. We fit T_N to a power $C_T N^\alpha$. In 2D, we expect $\alpha \approx 2$; in 3D, $\alpha \approx 3$. In 3D, we fit E_N to $C_E h^\beta$, and expect $\beta \approx -1$ in all cases, due to the use of local factoring. In fact, for `olim26` and `olim3d` using either `mp0` or `mp1`, we find that the situation is better than expected, with $\beta \approx -1.3$.

points, so that $h = (N-1)^{-1}$.

For this choice of slowness function, we plot the CPU runtime versus N (see figure 2.18), along with the relative ℓ_∞ error versus N (see figure 2.18). We also do least squares fits for these plots to get an overall sense of the accuracy and speed (see 2.1).

We can see that our conclusions from section 2.12 also hold for the multiple point source problem. Additionally, our least-squares fits (table 2.1) indicate to us that our algorithms' runtimes are accurately described by the fit $T_N \sim C_T N^\alpha$ with $\alpha \approx n$, and the error by $E_N \sim C_E h^\beta$, with $\beta \approx 1$ (here, E_N is the relative ℓ_∞ error). In fact, for `olim26` and `olim3d` with `mp0` or `mp1`, the power β is improved beyond 1 to $\beta \approx 1.3$.

2.13 Conclusion

We have presented a family of fast and accurate direct solvers for the eikonal equation. The *top-down* algorithm relies on enumerating valid update simplexes, while the *bottom-up* algorithm employs a fast search for the first arrival characteristic. For each of these solvers, one can use different quadrature rules: a simplified midpoint

rule (`mp0`), a midpoint rule (`mp1`), and a righthand rule (`rhr`).

We have analyzed the relationship between these quadrature rules, showing that the `mp0` rule can be used to compute an approximate local characteristic direction, and $\hat{U}$ evaluated using this direction, while incurring only $O(h^3)$ error per update, which justifies its use.

We have conducted extensive numerical experiments that show that `olim3d_mp0` provides the best overall trade-off between runtime and error. We also compare the speed of the standard fast marching method in 3D with the equivalent `olim6_rhr` (equivalent in the sense that they compute the same solution to machine precision). We demonstrate that `olim6_rhr` incurs only a very modest overhead, suggesting that the *top-down* approach is an efficient way of generalizing the fast marching method; it also suggests that the *bottom-up* approach is a viable approach to speeding up Dijkstra-like algorithms in 3D, and should be viable for other types of algorithms that solve related equations (indeed, this has already been demonstrated for the quasipotential [140]).

To determine the relative time spent on different tasks, we have profiled our C++ implementation using Valgrind, separating time spent into several coarse-grained categories. From this, we show that for practical problem sizes, the runtime of Dijkstra-like algorithms behaves like CN^n, where $n = 2, 3$, and N^n is the total number of gridpoints (even if this is not strictly true from a computational complexity viewpoint); we also emphasize that memory access patterns play a large role in algorithm runtime, especially for large N.

We conclude that ordered line integral methods are a powerful approach to obtaining a higher degree of accuracy when solving the eikonal equation in 3D. With an appropriate choice of quadrature rule, we are able to exploit improved directional coverage to drive down the error constant. The improved accuracy more than makes up for the modest price paid in speed, and we fully expect it to be possible to find ways

to optimize this family of algorithms further. We have also attempted to demonstrate that memory access patterns dominate both update time and time spent maintaining the front data structure, from which we can conclude two things: 1) the exact time spent updating a node is important but not paramount (improving accuracy is more important than improving speed), 2) using memory optimally will lead to a substantial speed-up for large problems.

Chapter 3

The jet marching method in 2D

3.1 Introduction

Our goal is to develop a family of high-order semi-Lagrangian eikonal solvers which use compact stencils. This is motivated by problems in high-frequency room acoustics, although the eikonal equation arises in a tremendous variety of modeling problems [118].

In multimedia, virtual reality, and video games, precomputing room impulse responses (RIRs) or transfer functions (RTFs) enables convincing spatialized audio, in combination with binaural or surround sound formats. Such an approach, usually referred to as numerical acoustics, involves computing pairs of RIRs by placing probes at different locations in a voxelized domain, numerically solving the acoustic wave equation, and capturing salient perceptual parameters throughout the domain using a streaming encoder [109, 110]. These parameters are later decoded using signal processing techniques in real time as the listener moves throughout the virtual environment. Assuming that the encoded parameters can comfortably fit into memory, a drawback of this approach is that the complexity of the simulation depends intrinsically on the highest frequency simulated. In practice, simulations top out at around

1 kHz. The hearing range of humans is roughly 20 Hz to 20 kHz, which requires these methods to either implicitly or explicitly extrapolate the bandlimited transfer functions to the full audible spectrum.

An established alternative to this approach is geometric acoustics, where methods based on raytracing are used [112]. Contrary to methods familiar from computer graphics, the focus of geometric acoustics is different. Acoustic waves are mechanical and have macroscopic wavelengths. This means that subsurface scattering, typically modeled using BRDFs in raytracing for computer graphics [93], is less relevant, and is limited to modeling macroscopic scattering from small geometric features, since reflections from flat surfaces are specular in nature. What's more, accurately modeling diffraction effects is crucial [113]: e.g., we can hear a sound source occluded by an obstacle, but we can't see it. A variety of other geometric-acoustic methods exist beyond raytracing. Examples include the image source method [5] and frustum tracing [33].

Geometric acoustics and optics both assume a solution to the wave equation based on an asymptotic high-frequency (WKB) approximation to the Helmholtz equation [97]. In this approximation, the eikonal plays the role of a spatially varying phase function, whose level sets describe propagating wavefronts. The prefactor of this approximation describes the amplitude of these wavefronts. The WKB approximation assumes a ray of "infinite frequency", suitable for optics, since the effects of diffraction are limited. A variety of mechanisms for augmenting this approximation with frequency-dependent diffraction effects have been proposed, the most successful of which is Keller's geometric theory of diffraction [69] (including the later uniform theory of diffraction [75]).

The complete geometric acoustic field of multiply reflected and diffracted rays can be parametrized by repeatedly solving the eikonal equation, using boundary conditions derived from the WKB approximation to patch together successive fields. A

related approach is Benamou's big raytracing (BRT) [12, 13]. This approach requires one to be able to accurately solve the transport equation describing the amplitude, e.g. using paraxial raytracing [97]. In order to do this, the first and second order partial derivatives of the eikonal must be computed. High-order accurate iterative schemes for solving the eikonal equation exist [142, 139, 82], but their performance deteriorates in the presence of complicated obstacles. Direct solvers for the eikonal equation allow one to locally parametrize the characteristics (rays) of the eikonal equation, which puts one in a position to simultaneously march the amplitude. This enables the design of work-efficient algorithms, critical if a large number of eikonal problems must be solved.

Benamou's line of research related to BRT seems to have stalled due to difficulties faced with caustics [15]. This is reasonable considering that the intended use was seismic modeling, where the eikonal equation is used to model first arrival times of P-waves. In this case, the speed of sound is extremely complicated, resulting in a large number of caustics [136]. On the other hand, in room acoustics, the speed of sound varies slowly. The main challenge is geometric: the domain is potentially filled with obstacles. This provides another motivation for compact stencils: such stencils can be adapted for use with unstructured meshes, and the sort of complicated boundary conditions that arise when using finite differences are avoided entirely.

The solvers developed in this work are high-order, have optimally local/compact stencils, and are label-setting methods (much like Sethian's fast marching method [117] or Tsitsiklis's semi-Lagrangian algorithm for solving the eikonal equation [134]). Additionally, being semi-Lagrangian, they locally parametrize characteristics (acoustic rays), making them suitable for use with paraxial raytracing [97], the method of choice for locally computing the amplitude. To the best of our knowledge, these are the first eikonal solvers with this collection of properties.

We refer to our solvers as jet marching methods to reflect the fact that the key

idea is marching the jet of the eikonal in a principled fashion. The jet of a function is a vector of the coefficients of a truncated Taylor polynomial approximating that function at a fixed base point[121]. For example, we could think of $(\tau(\boldsymbol{x}), \nabla\tau(\boldsymbol{x}), \nabla^2\tau(\boldsymbol{x}))$ as a particular jet of τ at the point $\boldsymbol{x}$. Since the jet provides the data required for Hermite interpolation of a certain order, marching jets causally will allow us to do Hermite interpolation locally for each semi-Lagrangian update.

Sethian and Vladimirsky developed a fast marching method that additionally marched the gradient of the eikonal in a short note, but did not prove convergence results or provide detailed numerical experiments [119]. Related methods exist in the level set method community and are referred to as gradient-augmented level set methods or jet schemes [90, 115].

In the rest of this work we lay out these methods, provide detailed numerical experiments, and give some preliminary theoretical performance guarantees. Our presentation is for unstructured grids in n-dimensions, while our numerical experiments were carried out in 2D. We plan to extend these solvers to structured and unstructured meshes in 3D and will report on these later in the context of room acoustics applications.

Problem setup

Let $\Omega \subseteq \mathbb{R}^n$ be a domain, let $\partial\Omega$ be its boundary, and let $\Gamma \subseteq \Omega$. The eikonal equation is a nonlinear first-order hyperbolic partial differential equation given by:

$$\begin{aligned} \|\nabla\tau(\boldsymbol{x})\| &= s(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega, \\ \tau(\boldsymbol{x}) &= g(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Gamma. \end{aligned} \tag{3.1}$$

Here, $\tau : \Omega \to \mathbb{R}$ is the eikonal, a spatial phase function that encodes the first arrival time of a wavefront propagating with pointwise slowness specified by $s : \Omega \to (0, \infty)$, which can be thought of as an index of refraction. The function $g : \Gamma \to \mathbb{R}$ specifies the boundary conditions, and is subject to certain compatibility conditions [22].

One way of arriving at the eikonal equation is by approximating the solution u of the Helmholtz equation:

$$\Big(\Delta + \omega^2 s(\boldsymbol{x})^2\Big) u(\boldsymbol{x}) = 0, \tag{3.2}$$

with the WKB ansatz:

$$u(\boldsymbol{x}) \sim \alpha(\boldsymbol{x}) e^{i\omega\tau(\boldsymbol{x})}, \tag{3.3}$$

where ω is the frequency [97]. As $\omega \to \infty$, this asymptotic approximation is $O(\omega^{-1})$ accurate. This is referred to as the geometric optics approximation [15]. The level sets of τ denote the arrival times of bundles of rays, and the amplitude α, which satisfies the transport equation:

$$\alpha(\boldsymbol{x})\Delta\tau(\boldsymbol{x}) + 2\nabla\tau(\boldsymbol{x})^\top \nabla\alpha(\boldsymbol{x}) = 0, \tag{3.4}$$

describes the attenuation of the amplitude of the wavefront due to the propagation and geometric spreading of rays. The characteristics of the eikonal equation satisfy the raytracing ODEs.

The solution of the eikonal equation can also be described using Fermat's principle:

$$\tau(\boldsymbol{x}) = \min_{\substack{\boldsymbol{y}\in\Gamma \\ \boldsymbol{\psi}:[0,1]\to\Omega \\ \boldsymbol{\psi}(0)=\boldsymbol{y},\boldsymbol{\psi}(1)=\boldsymbol{x}}} \left\{ \tau(\boldsymbol{y}) + \int_0^1 s(\boldsymbol{\psi}(\sigma)) \|\boldsymbol{\psi}'(\sigma)\| d\sigma \right\}. \tag{3.5}$$

Observe that this equation is recursive, suggesting a connection with dynamic programming and Bellman's principle of optimality. Indeed, the path $\boldsymbol{\psi}$ is a ray whose Lagrangian and Hamiltonian are:

$$\mathcal{H}(\boldsymbol{x}, \nabla\tau(\boldsymbol{x})) = \frac{\|\nabla\tau(\boldsymbol{x})\|^2 - s(\boldsymbol{x})^2}{2} = 0, \qquad \mathcal{L}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = s(\boldsymbol{x})\|\dot{\boldsymbol{x}}\|. \tag{3.6}$$

This provides the connection between the Eulerian perspective given by the eikonal equation, Fermat's principle, and the Lagrangian view provided by raytracing.

3.2 Related work

The quintessential numerical method for solving the eikonal equation is the fast marching method [116]. We discretize Ω into a grid of nodes Ω_h, where $h > 0$ is the characteristic length scale of elements in Ω_h. Let $T : \Omega_h \to \mathbb{R}$ be the numerical eikonal. To compute T, equation (1.4) is discretized using first-order finite differences and the order in which individual values of T are relaxed is determined using a variation of Dijkstra's algorithm for solving the single source shortest paths problem [117, 118]. If $N = |\Omega_h|$, then the fast marching method solves (1.4) in $O(N \log N)$ with $O(h \log \frac{1}{h})$ worst-case accuracy [143]. The logarithmic factor only appears when rarefaction fans are present: e.g., point source boundary data, or if the wavefront diffracts around a singular corner or edge. In these cases, full $O(h)$ accuracy can be recovered by proper initialization near rarefaction fans, or by employing a variety of factoring schemes [52, 82, 106].

It is also possible to solve the eikonal equation using semi-Lagrangian numerical methods, in which the ansatz (3.5) is discretized and applied locally [134, 101]. For instance, at a point $\hat{\boldsymbol{x}} \in \Omega_h$, we consider a neighborhood of points $\mathtt{nb}(\boldsymbol{x}) \subseteq \Omega_h$, assume that τ is fixed over the "surface" of this neighborhood, and approximate (3.5). As an example, if $\mathtt{nb}(\hat{\boldsymbol{x}})$ consists of its $2n$ nearest neighbors, if we linearly interpolate τ over the facets of $\mathrm{conv}(\mathtt{nb}(\hat{\boldsymbol{x}}))$, and discretize the integral in (3.5) using a right-hand rule, the resulting solver is equivalent to the fast marching method [120].

The Eulerian approach has generally been favored when developing higher-order solvers for the eikonal equation [142]. The eikonal equation is discretized using higher-order finite difference schemes and solved in the same manner as the fast marching method or using a variety of appropriate iterative schemes. Unfortunately, these approaches presuppose a regular grid and require wide stencils.

Our goal is to develop solvers for the eikonal equation that are high order, are optimally local (only use information from the nodes in $\mathtt{nb}(\hat{\boldsymbol{x}})$ to update $\hat{\boldsymbol{x}}$), and are

flexible enough to work on unstructured meshes. Using a semi-Lagrangian approach based on a high-order discretization of (3.5) allows us to do this.

This work was inspired by several lines of research. First, are gradient-augmented level set methods (or jet schemes) [90, 115]. Although developed for solving time-dependent advection problems, trying to map ideas from the time-dependent to the time-independent (or static) setting is natural, and presented an intriguing challenge. Second, the idea of using a semi-Lagrangian solver to construct a finite element solution to the eikonal equation incrementally was informative [22]; while the authors only constructed a first-order finte element approximation, attempting to push past this formulation to obtain a higher-order solver is a natural extension. Third, we were motivated by Chopp's idea of building up piecewise bicubic interpolants locally while marching the eikonal [35]; indeed, Chopp's work is mentioned in the original work on jet schemes in a similar capacity.

3.3 The jet marching method

Label-setting algorithms [29], such as the fast marching method, compute an approximation to τ by marching a numerical approximation $T : \Omega_h \to \mathbb{R}^n$ throughout the domain. The boundary data g is not always specified at the nodes of Ω_h. Let Γ_h be a discrete approximation of Γ. Once T is computed at $\Gamma_h \subseteq \Omega_h$ with sufficiently high accuracy, the solver begins to operate. To drive the solver, a set of states $\{\texttt{far}, \texttt{trial}, \texttt{valid}\}$ is used for bookkeeping. We initially set:

$$\texttt{state}(\boldsymbol{x}) = \begin{cases} \texttt{trial}, & \text{if } \boldsymbol{x} \in \Gamma_h, \\ \texttt{far}, & \text{otherwise.} \end{cases} \tag{3.7}$$

The `trial` nodes are typically sorted by their T value into an array-based binary heap implementing a priority queue, although alternatives have been explored [58]. At each step of the iteration, the node $\boldsymbol{x}$ with the minimum T value is removed from

the heap, $\texttt{state}(\boldsymbol{x})$ is set to `valid`, the `far` nodes in $\texttt{nb}(\boldsymbol{x})$ have their state set to `trial`, and each `trial` node in $\texttt{nb}(\boldsymbol{x})$ is subsequently updated. We have additionally provided a video online which shows the algorithm running [27].

From this, we can see that the value $T(\boldsymbol{x})$ depends on the values of T at the nodes of a directed graph leading from $\boldsymbol{x}$ back to Γ_h, noting that $T(\boldsymbol{x})$ can—and in general does—depend on multiple nodes in $\texttt{nb}(\boldsymbol{x})$. This means that the error in T accumulates as the solution propagates downwind from Γ_h. We generally assume that the depth of the directed graph of updates connecting each $\boldsymbol{x} \in \Omega_h$ to Γ_h is $O(h^{-1})$. The error due to each update comes from two sources: the running error accumulated in T, and the error incurred by approximating the integral in (3.5). For this reason, we would expect the order of the global error of the solver to be one less than the local error. However, the situation is more complicated, and the numerical analysis is lengthy, complicated, and nuanced. We state some preliminary theoretical results for a particular JMM in section 3.9.

Regardless, we assume that we only know the values of the eikonal and some of its derivatives at the nodes $\boldsymbol{x} \in \Omega_h$. To obtain higher-order accuracy locally, we make use of piecewise Hermite elements. In particular, at each node $\boldsymbol{x}$, we approximate the jet of the eikonal; i.e., τ and a number of its derivatives [121]. If we compute the jet with sufficiently high accuracy when we set $\texttt{state}(\boldsymbol{x}) \leftarrow \texttt{valid}$, we will be in a position to approximate τ using Hermite interpolation locally over $\operatorname{conv}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$. We consider several variations on this idea.

The general cost function

Fix a point $\hat{\boldsymbol{x}} \in \Omega_h$, thinking of it as the update point. To compute $T(\hat{\boldsymbol{x}})$, we consider sets of `valid` nodes $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d\} \subseteq \texttt{nb}(\hat{\boldsymbol{x}})$, where $1 \leq d \leq n$. The tuple of nodes $(\hat{\boldsymbol{x}}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_d)$ is an update of dimension d, and the collection of updates a stencil. We refer to the nodes $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d\}$ as the vertices of the base of the update.

In some cases, such as on an unstructured mesh, stencils may vary with $\hat{\boldsymbol{x}}$.

Necessary conditions on the updates and stencils for monotonic convergence have begun to be studied, and come in the form of causality conditions [72]. In particular, the cone spanned by $\{\boldsymbol{x}_1 - \hat{\boldsymbol{x}}, \ldots, \boldsymbol{x}_n - \hat{\boldsymbol{x}}\}$ should fit inside the nonnegative orthant after being rotated [119, 120]. It is not clear that causal stencils are also sufficient for monotone convergence. It appears to be necessary for the union of the cones spanned by each update to cover $\mathbb{R}^n$, but this union need not partition $\mathbb{R}^n$. Furthermore, for $O(h)$ solvers that do not make use of gradient information, a variety of stencils lead to monotone convergence. However, in section 3.9, we present a simple counterexample leading to a reduced order of convergence.

In previous works, we have explored a variety of ways of building stencils, but a simple approach is to mesh the surface of $\mathrm{conv}(\mathtt{nb}(\hat{\boldsymbol{x}}))$, letting the stencil be comprised of the faces of the mesh [101, 140].

To describe a general update, without loss of generality we assume $d = n$, and assume that the update nodes are in general position. That is, if we choose n nodes from $\{\hat{\boldsymbol{x}}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, the remaining node does not lie in their convex hull. We assume that we have access to a sufficiently accurate approximation of τ over $\mathrm{conv}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, call it T. We distinguish between T and T in the following way: T denotes the local numerical approximation of τ used by a particular update, while T denotes the global numerical approximation of τ. The two may not be equal to each other. Indeed, T is in general only defined on Ω_h, while T is only defined on $\mathrm{conv}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.

Let $\boldsymbol{x}_{\boldsymbol{\lambda}} \in \mathrm{conv}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, and let $L = L_{\boldsymbol{\lambda}} = \|\hat{\boldsymbol{x}} - \boldsymbol{x}_{\boldsymbol{\lambda}}\|$. Recall that $\boldsymbol{\psi} : [0, L] \to \Omega$ is the curve minimizing (3.5) for a particular choice of $\boldsymbol{x}_{\boldsymbol{\lambda}}$. We approximate $\boldsymbol{\psi}$ with a cubic parametric curve $\boldsymbol{\varphi} : [0, L] \to \Omega$ such that:

$$\boldsymbol{\varphi}(0) = \boldsymbol{x}_{\boldsymbol{\lambda}}, \qquad \boldsymbol{\varphi}(L) = \hat{\boldsymbol{x}}, \qquad \boldsymbol{\varphi}'(0) \sim \boldsymbol{t}_{\boldsymbol{\lambda}}, \qquad \boldsymbol{\varphi}'(L) \sim \hat{\boldsymbol{t}}, \tag{3.8}$$

and where $\boldsymbol{t}_{\boldsymbol{\lambda}}$ and $\hat{\boldsymbol{t}}$ are tangent vectors which enter as parameters. Note that $\boldsymbol{\varphi}'(0)$ and $\boldsymbol{\varphi}'(L)$ may not be exactly equal to $\boldsymbol{t}_{\boldsymbol{\lambda}}$ and $\hat{\boldsymbol{t}}$.

We approximate the integral in (3.5) over $\boldsymbol{\varphi}$ using Simpson's rule. This gives the cost functional:

$$F(\boldsymbol{\varphi}) = \mathsf{T}(\boldsymbol{x}_{\boldsymbol{\lambda}}) + \frac{L}{6}\Big[s(\boldsymbol{x}_{\boldsymbol{\lambda}})\|\boldsymbol{\varphi}'(0)\| + 4s\big(\boldsymbol{\varphi}_{1/2}\big)\big\|\boldsymbol{\varphi}'_{1/2}\big\| + s(\hat{\boldsymbol{x}})\|\boldsymbol{\varphi}'(L)\|\Big], \tag{3.9}$$

where $\boldsymbol{\varphi}_{1/2} = \boldsymbol{\varphi}(L/2)$ and $\boldsymbol{\varphi}'_{1/2} = \boldsymbol{\varphi}'(L/2)$. We have not yet made this well-defined. To do so, we must specify T, $\boldsymbol{t}_{\boldsymbol{\lambda}}$, and $\hat{\boldsymbol{t}}$. We describe several different ways of doing this in the following sections.

Computing $\nabla T(\hat{\boldsymbol{x}})$

A minimizing extremal $\boldsymbol{\psi}$ of Fermat's integral is a characteristic of the eikonal equation. A simple but important consequence of this is that its tangent vector is locally parallel to $\nabla\tau$. Hence:

$$s(\boldsymbol{\psi}(\sigma))\frac{\boldsymbol{\psi}'(\sigma)}{\|\boldsymbol{\psi}'(\sigma)\|} = \nabla\tau(\boldsymbol{\psi}(\sigma)). \tag{3.10}$$

After minimizing F, we will have found an optimal value of $\hat{\boldsymbol{t}}$. We can then set:

$$\nabla T(\hat{\boldsymbol{x}}) \leftarrow s(\hat{\boldsymbol{x}})\hat{\boldsymbol{t}}. \tag{3.11}$$

This puts us in a position to march the gradient of the eikonal locally along with the eikonal itself.

Parametrizing $\boldsymbol{\varphi}$

We consider two methods of choosing $\boldsymbol{\varphi}$ (see Figure 3.1). Define:

$$\boldsymbol{\ell}(\sigma) = \boldsymbol{x}_{\boldsymbol{\lambda}} + \sigma\boldsymbol{\ell}', \qquad \boldsymbol{\ell}' = \frac{\hat{\boldsymbol{x}} - \boldsymbol{x}_{\boldsymbol{\lambda}}}{L_{\boldsymbol{\lambda}}}. \tag{3.12}$$

Here, $\boldsymbol{\ell}$ is the arc length parametrized straight line running from $\boldsymbol{x}_{\boldsymbol{\lambda}}$ to $\hat{\boldsymbol{x}}$, and $\boldsymbol{\ell}'$ is its unit tangent vector.

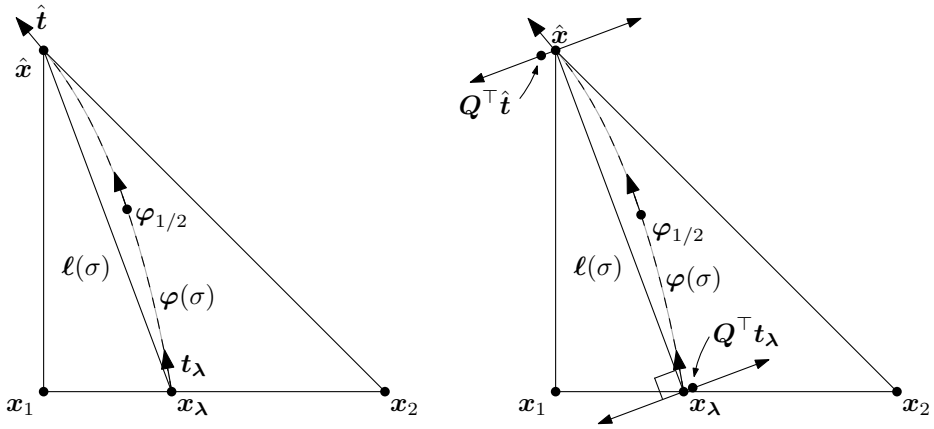


Figure 3.1: Two approaches to parametrizing a cubic curve approximating the characteristic $\boldsymbol{\varphi}$ leading from $\boldsymbol{x}_{\boldsymbol{\lambda}}$ to $\hat{\boldsymbol{x}}$ when numerically minimizing Fermat's integral to compute $T(\hat{\boldsymbol{x}})$ and $\nabla T(\hat{\boldsymbol{x}})$. *Left*: $\boldsymbol{\varphi}$ is a cubic parametric curve with boundary data set directly from $\boldsymbol{x}_{\boldsymbol{\lambda}}, \hat{\boldsymbol{x}}, \boldsymbol{t}_{\boldsymbol{\lambda}}$, and $\hat{\boldsymbol{t}}$. *Right*: $\boldsymbol{\varphi}$ is the graph of a function in the orthogonal complement of range$(\boldsymbol{\ell}')$.

Using a cubic parametric curve For one approach, we define:

$$\boldsymbol{\varphi}(\sigma) = \boldsymbol{\ell}(\sigma) + \delta\boldsymbol{\varphi}(\sigma), \tag{3.13}$$

where $\delta\boldsymbol{\varphi} : [0, L] \to \Omega$ is a perturbation away from $\boldsymbol{\ell}$ that satisfies:

$$\delta\boldsymbol{\varphi}(0) = \boldsymbol{x}_{\boldsymbol{\lambda}}, \quad \delta\boldsymbol{\varphi}(L) = \hat{\boldsymbol{x}}, \quad \delta\boldsymbol{\varphi}'(0) = \boldsymbol{t}_{\boldsymbol{\lambda}} - \boldsymbol{\ell}', \quad \delta\boldsymbol{\varphi}'(L) = \hat{\boldsymbol{t}} - \boldsymbol{\ell}'. \tag{3.14}$$

We can explicitly write $\delta\boldsymbol{\varphi}$ as:

$$\delta\boldsymbol{\varphi}(\sigma) = \big(\boldsymbol{t}_{\boldsymbol{\lambda}} - \boldsymbol{\ell}'\big)K_0(\sigma) + \big(\hat{\boldsymbol{t}} - \boldsymbol{\ell}'\big)K_1(\sigma), \tag{3.15}$$

where $K_0, K_1 : [0, L] \to \mathbb{R}$ are Hermite basis functions such that:

$$\begin{aligned} K_0(0) &= 0 = K_0(L), \qquad & K_1(0) &= 0 = K_1(L), \\ K_0'(0) &= 1 = K_1'(L), \qquad & K_1'(0) &= 0 = K_0'(L). \end{aligned} \tag{3.16}$$

Explicitly, these are given by:

$$K_0(\sigma) = \sigma - 2\frac{\sigma^2}{L} + \frac{\sigma^3}{L^2}, \qquad K_1(\sigma) = \frac{-\sigma^2}{L} + \frac{\sigma^3}{L^2}. \tag{3.17}$$

Let $\boldsymbol{t}_{\boldsymbol{\lambda}}, \hat{\boldsymbol{t}} \in \mathbb{S}^{n-1}$ so that $\|\boldsymbol{t}_{\boldsymbol{\lambda}}\| = 1 = \|\hat{\boldsymbol{t}}\|$. As $L \to 0$, this results in a curve that is approximately parametrized by arc length: i.e., $\|\boldsymbol{\varphi}'(\sigma)\| \to 1$ for all σ such that

$0 \leq \sigma \leq L$ [51]. This simplifies the general cost function given by (3.9) to:

$$F(\boldsymbol{\varphi}) = \mathsf{T}(\boldsymbol{x}_{\boldsymbol{\lambda}}) + \frac{L}{6}\Big[s(\boldsymbol{x}_{\boldsymbol{\lambda}}) + 4s\big(\boldsymbol{\varphi}_{1/2}\big)\big\|\boldsymbol{\varphi}'_{1/2}\big\| + s(\hat{\boldsymbol{x}})\Big]. \tag{3.18}$$

Using (3.17), $\boldsymbol{\varphi}_{1/2}$ and $\boldsymbol{\varphi}'_{1/2}$ can be written:

$$\boldsymbol{\varphi}_{1/2} = \frac{\boldsymbol{x}_{\boldsymbol{\lambda}} + \hat{\boldsymbol{x}}}{2} + \frac{L}{8}\big(\boldsymbol{t}_{\boldsymbol{\lambda}} - \hat{\boldsymbol{t}}\big), \qquad \boldsymbol{\varphi}'_{1/2} = \frac{3}{2}\boldsymbol{\ell}' - \frac{\boldsymbol{t}_{\boldsymbol{\lambda}} + \hat{\boldsymbol{t}}}{4}. \tag{3.19}$$

Note that $\boldsymbol{\varphi}_{1/2} \sim (\boldsymbol{x}_{\boldsymbol{\lambda}} + \hat{\boldsymbol{x}})/2$ and $\boldsymbol{\varphi}'_{1/2} \sim \boldsymbol{\ell}'$ as $L \to 0$ if we assume that the wavefront is well-approximated by a plane wave near the update, since in this case $\boldsymbol{t}_{\boldsymbol{\lambda}} \sim \hat{\boldsymbol{t}} \sim \boldsymbol{\ell}'$.

Parametrizing $\boldsymbol{\varphi}$ as the graph of a function We can also define the perturbation away from $\boldsymbol{\ell}$ as the graph of a function; i.e., we assume that the perturbation is orthogonal to $\boldsymbol{\ell}'$. Letting $\boldsymbol{Q} \in \mathbb{R}^{n\times(n-1)}$ be an orthogonal matrix such that $\boldsymbol{Q}^\top \boldsymbol{\ell}' = 0$, and letting $\boldsymbol{\zeta} : [0, L] \to \mathbb{R}^{n-1}$ be a curve specifying the components of the perturbation in this basis, we choose $\delta\boldsymbol{\varphi}(\sigma) = \boldsymbol{Q}\boldsymbol{\zeta}(\sigma)$ so that:

$$\boldsymbol{\varphi}(\sigma) = \boldsymbol{\ell}(\sigma) + \boldsymbol{Q}\boldsymbol{\zeta}(\sigma). \tag{3.20}$$

where $\boldsymbol{\zeta}(\sigma) = \boldsymbol{b}_0 K_0(\sigma) + \boldsymbol{b}_1 K_1(\sigma)$. In this approach, instead of $\hat{\boldsymbol{t}}$ and $\boldsymbol{t}_{\boldsymbol{\lambda}}$, we optimize over $\boldsymbol{b}_0, \boldsymbol{b}_1 \in \mathbb{R}^{n-1}$. Now, noting that:

$$\|\boldsymbol{\varphi}'(\sigma)\| = \sqrt{\|\boldsymbol{\ell}'\|^2 + \|\boldsymbol{Q}\boldsymbol{\zeta}'(\sigma)\|^2} = \sqrt{1 + \|\boldsymbol{\zeta}'(\sigma)\|^2}, \tag{3.21}$$

we can write the cost functional F as:

$$\begin{aligned} F(\boldsymbol{\varphi}) = \mathsf{T}(\boldsymbol{x}_{\boldsymbol{\lambda}}) + \frac{L}{6}\Bigg[& s(\boldsymbol{x}_{\boldsymbol{\lambda}})\sqrt{1 + \|\boldsymbol{b}_0\|^2} \\ & + 4s(\boldsymbol{\varphi}_{1/2})\sqrt{1 + \|(\boldsymbol{b}_0 + \boldsymbol{b}_1)/4\|^2} + s(\hat{\boldsymbol{x}})\sqrt{1 + \|\boldsymbol{b}_1\|^2}\Bigg]. \end{aligned} \tag{3.22}$$

Trade-offs between the two parametrizations of $\boldsymbol{\varphi}$ When $\boldsymbol{\varphi}$ is a cubic parametric curve, we run into an interesting problem described in more detail by Floater [51]. In particular, the order of accuracy of $\boldsymbol{\varphi}$ in approximating $\boldsymbol{\psi}$ is limited by our

parametrization of $\boldsymbol{\varphi}$. If we parametrize $\boldsymbol{\varphi}$ over $\sigma \in [0,1]$ (that is, with a unit parametrization), then the interpolant is at most $O(h^2)$ accurate. If we parametrize it using a chordal parametrization, i.e. $\sigma \in [0, L]$, then it is at most $O(h^4)$ accurate. Indeed, any Hermite spline using a chordal parametrization over each of its segment is at most $O(h^4)$ accurate globally. To design a higher order solver than this requires us to parametrize $\boldsymbol{\varphi}$ using a more accurate approximation of the arc length of $\boldsymbol{\psi}$ (consider, e.g., using a quintic parametric curve). On the other hand, if we parametrize $\boldsymbol{\varphi}$ as the graph of a function, we can directly apply Hermite interpolation theory [127], and there is no such obstacle.

3.4 Different types of minimization problems

In this section, we consider four different ways of using F to pose a minimization problem which would allow us to compute $T(\hat{\boldsymbol{x}})$. We note that each of these formulations is compatible with the version of F where we take $\boldsymbol{\varphi}$ to be a parametric curve *and* where we define it as the graph of a function orthogonal to $\boldsymbol{\ell}(\sigma)$. Altogether, this leads to eight different JMMs.

Determining $\boldsymbol{t_\lambda}$ by minimizing Fermat's integral

Since the optimal $\boldsymbol{\varphi}$ is a characteristic of the eikonal equation, one approach to setting $\boldsymbol{t_\lambda}$ and $\hat{\boldsymbol{t}}$ is to simply let them enter into the cost function as free parameters to be optimized over. This leads to the optimization problem:

$$\begin{aligned} \text{minimize} \quad & F(\boldsymbol{x_\lambda}, \boldsymbol{t_\lambda}, \hat{\boldsymbol{t}}) \\ \text{subject to} \quad & \boldsymbol{x_\lambda} \in \operatorname{conv}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n), \\ & \boldsymbol{t_\lambda}, \hat{\boldsymbol{t}} \in \mathbb{S}^{n-1}, \end{aligned} \tag{3.23}$$

if we parametrize $\boldsymbol{\varphi}$ as a curve; or, if we parametrize $\boldsymbol{\varphi}$ as the graph of a function:

$$\begin{aligned} \text{minimize} \quad & F(\boldsymbol{x}_{\boldsymbol{\lambda}}, \boldsymbol{b}_0, \boldsymbol{b}_1) \\ \text{subject to} \quad & \boldsymbol{x}_{\boldsymbol{\lambda}} \in \operatorname{conv}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n), \\ & \boldsymbol{b}_0, \boldsymbol{b}_1 \in \mathbb{R}^{n-1}, \end{aligned} \tag{3.24}$$

For a d-dimensional update, the domain of each of these minimization problems has dimension $(d-1)(n-1)^2$, since $\dim(\operatorname{conv}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d)) = d-1$.

Determining $t_{\boldsymbol{\lambda}}$ from the eikonal equation

When we compute updates, we only require high-order accurate jets over $\operatorname{conv}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$. This is a subset of Ω of codimension one: an interval in 2D, or triangle in 3D. If we know T and ∇T at the vertices of this set, then we can use Hermite interpolation to compute T. Unfortunately, this means that we can only approximate directional derivatives of T in the linear span of this set. To compute $\nabla \mathsf{T}$, we need to recover the directional derivative normal to the facet.

Let $\tilde{\boldsymbol{V}} \in \mathbb{R}^{n \times (n-1)}$ be an orthogonal matrix such that:

$$\operatorname{range}(\tilde{\boldsymbol{V}}) = \operatorname{range}\left(\begin{bmatrix} \boldsymbol{x}_2 - \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_n - \boldsymbol{x}_1 \end{bmatrix}\right), \tag{3.25}$$

and let $\boldsymbol{v} \in \mathbb{R}^n$ be a unit vector such that $\tilde{\boldsymbol{V}}^\top \boldsymbol{v} = 0$. Let $\nabla_{\tilde{\boldsymbol{V}}}$ be the gradient restricted to the range of $\tilde{\boldsymbol{V}}$, and likewise let $d_{\boldsymbol{v}}$ denote the $\boldsymbol{v}$-directional derivative. Then, from the eikonal equation, we have:

$$s(\boldsymbol{x})^2 = \|\nabla \tau(\boldsymbol{x})\|^2 = |d_{\boldsymbol{v}} \tau(\boldsymbol{x})|^2 + \|\nabla_{\tilde{\boldsymbol{V}}} \tau(\boldsymbol{x})\|^2. \tag{3.26}$$

To recover $\nabla \tau(\boldsymbol{x})$, first note that $\nabla \tau(\boldsymbol{x})$ should point in the same direction as $\boldsymbol{\ell}'$. Choosing $\boldsymbol{v}$ so that $\boldsymbol{v}^\top \boldsymbol{\ell}' > 0$, we get:

$$d_{\boldsymbol{v}} \tau(\boldsymbol{x}) = \sqrt{s(\boldsymbol{x})^2 - \|\nabla_{\tilde{\boldsymbol{V}}} \tau(\boldsymbol{x})\|^2}. \tag{3.27}$$

Letting $\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v} & \tilde{\boldsymbol{V}} \end{bmatrix}$, equation (3.27) combined with $\nabla \tau(\boldsymbol{x}) = \boldsymbol{V} \nabla_{\boldsymbol{V}} \tau(\boldsymbol{x})$ gives us a means of recovering $\nabla \tau(\boldsymbol{x})$ from $\nabla_{\tilde{\boldsymbol{V}}} \tau(\boldsymbol{x})$.

Using this technique, we can pose the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & F(\boldsymbol{x}_{\boldsymbol{\lambda}}, \hat{\boldsymbol{t}}) \\ \text{subject to} \quad & \boldsymbol{x}_{\boldsymbol{\lambda}} \in \operatorname{conv}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n), \\ & \hat{\boldsymbol{t}} \in \mathbb{S}^{n-1}, \end{aligned} \tag{3.28}$$

or, optimizing over $\boldsymbol{b}_1$ directly:

$$\begin{aligned} \text{minimize} \quad & F(\boldsymbol{x}_{\boldsymbol{\lambda}}, \boldsymbol{b}_1) \\ \text{subject to} \quad & \boldsymbol{x}_{\boldsymbol{\lambda}} \in \operatorname{conv}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n), \\ & \boldsymbol{b}_1 \in \mathbb{R}^{n-1} \end{aligned} \tag{3.29}$$

For each $\boldsymbol{x}_{\boldsymbol{\lambda}}$, we set:

$$\boldsymbol{t}_{\boldsymbol{\lambda}} \leftarrow \frac{\boldsymbol{V}\nabla_{\boldsymbol{V}}\mathsf{T}(\boldsymbol{x}_{\boldsymbol{\lambda}})}{\|\boldsymbol{V}\nabla_{\boldsymbol{V}}\mathsf{T}(\boldsymbol{x}_{\boldsymbol{\lambda}})\|}. \tag{3.30}$$

The dimension of a d-dimensional update based on this minimization problem is $(d-1)(n-1)$

Determining $\boldsymbol{t}_{\boldsymbol{\lambda}}$ by marching cell-based interpolants

Another approach is to march cells that approximate the jet of the eikonal at each point. For example, if we have constructed a finite element interpolant using `valid` data on a cell whose boundary contains $\operatorname{conv}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ then we can evaluate its gradient to obtain:

$$\boldsymbol{t}_{\boldsymbol{\lambda}} \leftarrow \frac{\nabla\mathsf{T}(\boldsymbol{x}_{\boldsymbol{\lambda}})}{\|\nabla\mathsf{T}(\boldsymbol{x}_{\boldsymbol{\lambda}})\|}. \tag{3.31}$$

We can combine this approach with the cost functional given by (3.28), albeit with a modified $\boldsymbol{t}_{\boldsymbol{\lambda}}$. We elaborate on how we march cells in section 3.7. An advantage of this approach is that it allows one to simultaneously march the second partials of T.

A simplified method using a quadratic curve

In some cases, in particular if the speed of sound is linear, i.e.:

$$c(\boldsymbol{x}) = \frac{1}{s(\boldsymbol{x})} = c_0 + \boldsymbol{c}^{\top}\boldsymbol{x}, \qquad c_0 \in \mathbb{R}, \qquad \boldsymbol{c} \in \mathbb{R}^n, \tag{3.32}$$

the characteristic $\boldsymbol{\psi}$ is well-approximated by a quadratic. In this case, we again have a cost functional of the form (3.28).

If $\boldsymbol{\varphi}$ is parametrized as curve, we set $\boldsymbol{t}_{\boldsymbol{\lambda}}$ to be the reflection of $\hat{\boldsymbol{t}}$ across $\boldsymbol{\ell}'$:

$$\boldsymbol{t}_{\boldsymbol{\lambda}} = -\big(\boldsymbol{I} - 2\boldsymbol{\ell}'\boldsymbol{\ell}'^{\top}\big)\hat{\boldsymbol{t}}, \tag{3.33}$$

giving $\boldsymbol{t}_{\boldsymbol{\lambda}} + \hat{\boldsymbol{t}} = 2\boldsymbol{\ell}'\boldsymbol{\ell}'^{\top}\hat{\boldsymbol{t}}$ and $\boldsymbol{t}_{\boldsymbol{\lambda}} - \hat{\boldsymbol{t}} = -2\big(\boldsymbol{I} - \boldsymbol{\ell}'\boldsymbol{\ell}'^{\top}\big)\hat{\boldsymbol{t}}$. Then:

$$\boldsymbol{\varphi}_{1/2} = \frac{\boldsymbol{x}_{\boldsymbol{\lambda}} + \hat{\boldsymbol{x}}}{2} - \frac{L}{4}\big(\boldsymbol{I} - \boldsymbol{\ell}'\boldsymbol{\ell}'^{\top}\big)\hat{\boldsymbol{t}}, \qquad \boldsymbol{\varphi}'_{1/2} = \frac{3 + \boldsymbol{\ell}'^{\top}\hat{\boldsymbol{t}}}{2}\boldsymbol{\ell}'. \tag{3.34}$$

This simplifies F given by (3.18) to:

$$F(\boldsymbol{x}_{\boldsymbol{\lambda}}, \hat{\boldsymbol{t}}) = \mathsf{T}(\boldsymbol{x}_{\boldsymbol{\lambda}}) + \frac{L}{6}\left[s(\boldsymbol{x}_{\boldsymbol{\lambda}}) + 2\big(3 + \boldsymbol{\ell}'^{\top}\hat{\boldsymbol{t}}\big)s\Big(\frac{\boldsymbol{x}_{\boldsymbol{\lambda}} + \hat{\boldsymbol{x}}}{2} - \frac{L}{4}\big(\boldsymbol{I} - \boldsymbol{\ell}'\boldsymbol{\ell}'^{\top}\big)\Big) + s(\hat{\boldsymbol{x}})\right]. \tag{3.35}$$

If $\boldsymbol{\varphi}$ is parametrized as the graph of a function, then:

$$\boldsymbol{\zeta}_{1/2} = \frac{\hat{\boldsymbol{x}} + \boldsymbol{x}_{\boldsymbol{\lambda}}}{2} + \frac{L}{4}\boldsymbol{Q}^{\top}\hat{\boldsymbol{t}}, \qquad \boldsymbol{\zeta}' = 0, \tag{3.36}$$

simplifying the version of F in (3.22) to:

$$F(\boldsymbol{x}_{\boldsymbol{\lambda}}, \hat{\boldsymbol{t}}) = \mathsf{T}(\boldsymbol{x}_{\boldsymbol{\lambda}}) + \frac{L}{6}\left[\big(s(\boldsymbol{x}_{\boldsymbol{\lambda}}) + s(\hat{\boldsymbol{x}})\big)\sqrt{1 + \|\boldsymbol{b}_0\|^2} + 4s(\boldsymbol{\varphi}_{1/2})\right]. \tag{3.37}$$

since $\boldsymbol{Q}^{\top}\hat{\boldsymbol{t}} = \boldsymbol{b}_0 = -\boldsymbol{b}_1$.

Other approaches

We tried two other approaches which failed to provide satisfactory results:

- A combination of the quadratic simplification in subsection 3.4 with the methods in subsections 3.4 or 3.4. In this case, we use our knowledge of $\nabla T(\boldsymbol{x}_{\boldsymbol{\lambda}})$ along the base of the update to choose $\hat{\boldsymbol{t}}$ and $\boldsymbol{t}_{\boldsymbol{\lambda}}$. This reduces the dimensionality of the cost function to $d - 1$. However, except for in special cases (e.g. $s \equiv 1$), this propagates errors in a manner that causes the solver to diverge; or, at best, allows it converge with $O(h)$ accuracy. We note that if $s \equiv 1$, still simpler methods can be used, so this combination of approaches does not seem to be useful.

- We can extract not only $\boldsymbol{t}_{\boldsymbol{\lambda}}$ from the Hermite interpolant on $\operatorname{conv}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, but also $\boldsymbol{\varphi}''(0)$. From the Euler-Lagrange equations for the eikonal equation, we obtain:

$$\boldsymbol{Q}^\top \nabla s(\boldsymbol{x}_{\boldsymbol{\lambda}}) = s(\boldsymbol{x}_{\boldsymbol{\lambda}}) \frac{\boldsymbol{Q}^\top \boldsymbol{\varphi}''(0)}{\boldsymbol{\varphi}'(0)}. \tag{3.38}$$

  With $\boldsymbol{\varphi}$ parametrized as a graph, we have $\boldsymbol{Q}^\top \boldsymbol{\varphi}''(0) = \boldsymbol{\zeta}''(0)$, giving:

$$\boldsymbol{\zeta}''(0) = \frac{\boldsymbol{Q}^\top \nabla s(\boldsymbol{x}_{\boldsymbol{\lambda}})(1 + \|\boldsymbol{Q}^\top \boldsymbol{x}_{\boldsymbol{\lambda}}\|^2)}{s(\boldsymbol{x}_{\boldsymbol{\lambda}})}. \tag{3.39}$$

  This completely defines $\boldsymbol{\varphi}$ as the graph of a cubic polynomial using the graph parametrization. Unfortunately, this method diverges for the same reason as the method described in the previous bullet.

Optimization algorithms

We do not dwell on the details of how to numerically solve the minimization problems in the preceding sections. We make some general observations:

- These optimization problems are very easy to solve—what's costly is that we have to solve $O(N)$ of them. As $h \to 0$, they are strictly convex and well-behaved. Empirically, Newton's method converges in $O(1)$ steps (typically fewer than 5 with a well-chosen warm start). We leave a detailed comparison of different approaches to numerically solving these optimization problems for future work.

- The gradients and Hessians of these cost functions are somewhat complicated. Programming them can be tricky and tedious, suggesting that automatic differentiation may be a worthwhile approach [91, 59].

- The constraint $\boldsymbol{x}_{\boldsymbol{\lambda}} \in \operatorname{conv}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ corresponds to a set of linear inequality constraints, which are simple to incorporate. Because of the form of these con-

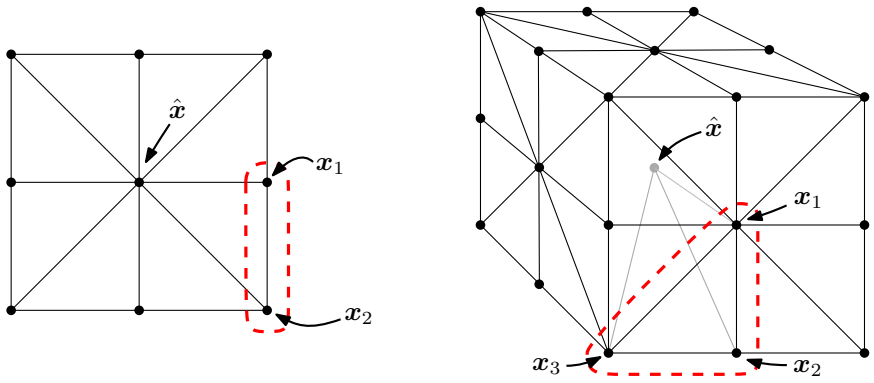
straints, checking the KKT conditions at the boundary is cheap and easy [101, 140]. See the next section on skipping updates.

- The constraints $\boldsymbol{t}_{\boldsymbol{\lambda}}, \hat{\boldsymbol{t}} \in \mathbb{S}^{n-1}$ are nonlinear equality constraints. However, these constraints can be eliminated. If $n = 2$, then we can set $\hat{\boldsymbol{t}} = (\cos(\theta), \sin(\theta))$, letting $\theta \in \mathbb{R}$. For $n > 2$, we can use a Riemannian Newton's method for minimization on $\mathbb{S}^{n-1}$, which is simple to implement and known to converge superlinearly [1].

3.5 Hierarchical update algorithms

Away from shocks, where multiple wavefronts collide, exactly one characteristic will pass through a point $\hat{\boldsymbol{x}}$. When we minimize F over each update in the stencil, the characteristic will pass through the base of the minimizing update, or possibly through the boundary of several adjacent updates. We can use this fact to sequence the updates that are performed to design a work-efficient solver. In our previous work on OLIMs (see Chapter 2, we explored variations of this idea [101, 140]. An approach that works well is the bottom-up update algorithm.

To fix the idea in 3D, consider $\texttt{nb}(\boldsymbol{x})$ as shown in Figure 3.2, for which $|\texttt{nb}(\boldsymbol{x})| = 26$. There are 26 "line" updates, where $d = 1$. To start with, each `valid` line update is done, and $\boldsymbol{x}_1$ for the minimizing line update is recorded. Next, we fix $\boldsymbol{x}_1$ and perform "triangle" updates ($d = 2$) where $\boldsymbol{x}_2$ is varying. In this case, we can restrict the number of triangle updates that are done by assuming either that $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ is an edge of mesh discretizing the surface of the 3D stencil shown in Figure 3.2, or that $\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ is small enough (measuring the distance of these two points in different norms leads to a different number of triangle updates—we find the ℓ_1 norm to work well). Finally, we fix $\boldsymbol{x}_2$ corresponding to the minimizing triangle update, and do tetrahedron updates containing $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Throughout this process, $\boldsymbol{x}_1, \boldsymbol{x}_2$, and $\boldsymbol{x}_3$



$|\texttt{nb}(\hat{\boldsymbol{x}})| = 8$ $|\texttt{nb}(\hat{\boldsymbol{x}})| = 26$

Figure 3.2: The neighborhoods typically used by semi-Lagrangian solvers in 2D and 3D on a regular grid. These are the stencils used by Tsitsiklis's algorithm and two of the OLIM stencils [134, 101]. *Left*: "`olim8`" in $\mathbb{R}^2$. This is the 8-point stencil used in this paper. *Right*: "`olim26`" in $\mathbb{R}^3$.

must all be `valid`.

We emphasize that our work-efficient OLIM update algorithms work equally well for the class of algorithms developed here. The main differences between the JMMs studied here and the earlier OLIMs are the cost functionals and the we way approximate T.

3.6 Initialization methods

A common problem with the convergence of numerical methods for solving the eikonal equation concerns how to treat rarefaction fans. Our numerical tests consist of point source problems, around which a rarefaction forms. A standard approach is to introduce the factored eikonal equation [52, 82, 106]. If a point source is located at $\boldsymbol{x}^\circ \in \Omega_h$ and if we set $\Gamma_h = \{\boldsymbol{x}^\circ\}$, then we let $d(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{x}^\circ\|$ and use the ansatz:

$$\tau(\boldsymbol{x}) = z(\boldsymbol{x}) + d(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega. \tag{3.40}$$

We insert this into the eikonal equation, modifying our numerical methods as necessary, and solve for $z(\boldsymbol{x})$ instead. This is not complicated—see our previous work on

OLIMs for solving the eikonal equation to see how the cost functions should generally be modified [101, 140].

Yet another approach would be to solve the characteristic equations to high-order for each $\boldsymbol{x}$ in such a ball. This would require solving $O(N)$ boundary value problems, each discretized into $O(N^{1/3})$ intervals, resulting in an $O(N^{4/3})$ cost overall (albeit with a very small constant). One issue with this approach is that it only works well if the ball surrounding $\boldsymbol{x}^\circ$ is contained in the interior of Ω. For our numerical experiments, we simply initialize T and ∇T to the correct, ground truth values in a ball or box of constant size centered at $\boldsymbol{x}^\circ$.

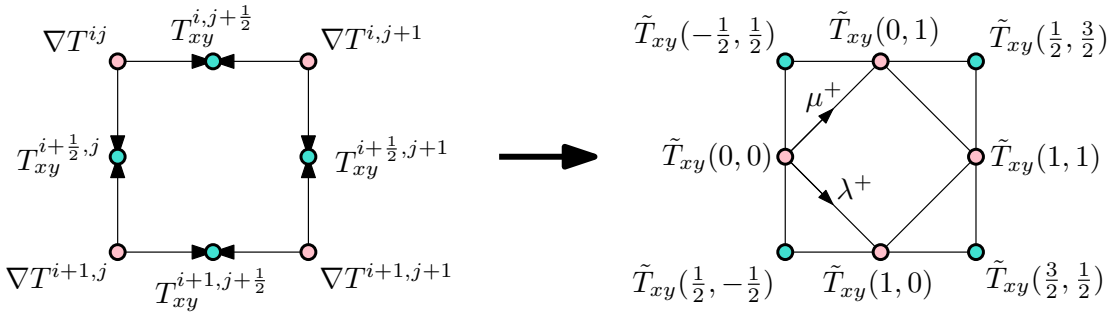
3.7 Cell marching

Of particular interest is solving the transport equation governing the amplitude α while simultaneously solving the eikonal equation. Equation (3.4) can be solved using upwind finite differences [12] or paraxial raytracing [97]. We prefer the latter approach since it can be done locally, using the characteristic path $\boldsymbol{\varphi}$ recovered when computing $T(\hat{\boldsymbol{x}})$. Either approach requires accurate second derivative information (we need ΔT for upwind finite differences, or $\nabla^2 T$ for paraxial raytracing).

For the purposes of explanation and our numerical tests, we consider a rectilinear grid with square cells in $\mathbb{R}^2$. On each cell, our goal is to build a bicubic interpolant, approximating $T(\boldsymbol{x})$. This requires knowing $T, \nabla T$, and T_{xy} at each cell corner. If we know these values with $O(h^{4-p})$ accuracy, where p is the order of the derivative, then the bicubic is $O(h^{4-p})$ accurate over the cell. So far, we have described an algorithm that marches T and ∇T, which together constitute the total 1-jet. We now show how T_{xy} can also be marched, allowing us to march the partial 1-jet.[1]

Let $\boldsymbol{x}_{ij}$ with $(i,j) \in \{0,1\}^2$ denote the corners of a square cell with sides of length

[1]The total k-jet of a function f is the set $\{\partial^{\boldsymbol{\alpha}} f\}_{\boldsymbol{\alpha}}$, where $\|\boldsymbol{\alpha}\|_1 \leq k$; the partial k-jet is $\{\partial^{\boldsymbol{\alpha}} f\}_{\boldsymbol{\alpha}}$ where $\|\boldsymbol{\alpha}\|_\infty \leq k$.



approximate T_{xy} at cell edge midpoints using central differences

approximate T_{xy} at cell vertices using bilinear extrapolation

Figure 3.3: *Cell-based interpolation.* To approximate the mixed second partials of a function with $O(h^2)$ accuracy from $O(h^3)$ accurate gradient values available at the corners of a cell, the following method of using central differences to approximate the mixed partials at the midpoints of the edges of the cell, followed by bilinear extrapolation, can be used.

h, and assume that we know $\nabla T(\boldsymbol{x}_{ij})$ with $O(h^3)$ accuracy. We can use the following approach to estimate $T_{xy}(\boldsymbol{x}_{ij})$ at each corner:

- First, at the midpoints of the edges oriented in the x direction (resp., y direction), approximate T_{xy} using the central differences involving T_y (resp., T_x) at the endpoints. This approximation is $O(h^2)$ accurate at the midpoints.
- Use bilinear extrapolation to reevaluate T_{xy} at the corners of the cell, yielding $T_{xy}(\boldsymbol{x}_{ij})$, also with $O(h^2)$ accuracy.

This procedure is illustrated in Figure 3.3.

One issue with this approach is that it results in a piecewise interpolant that is only C^1 globally. That is, if we estimate the value of T_{xy} at a corner from each of the cells which are incident upon it, we will get different values in general. To compute a globally C^2 piecewise interpolant, we can average T_{xy} values over incident `valid` cells, where we define a `valid` cell to be a cell whose vertices are all `valid`. How to do this is shown in Figure 3.4.

The idea of approximating the partial 1-jet from the total 1-jet in an optimally

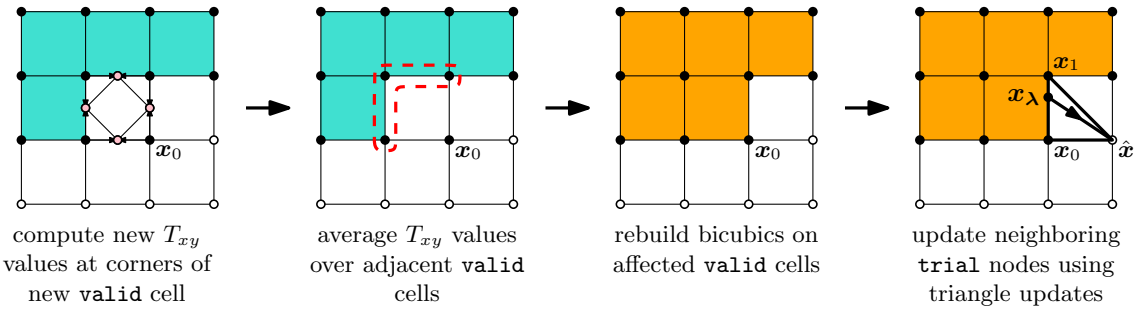


Figure 3.4: *Local cell marching.* After computing values of T_{xy} as shown in Figure 3.3 (*left*), to ensure continuity of the global interpolant, nodal values incident on the newly `valid` cell (containing x_0) can be recomputed by averaging over T_{xy} values taken from incident `valid` cells (*middle*). Finally, a bicubic cell-based interpolant is constructed (*right*).

local fashion by combining central differences with bilinear extrapolation, and averaging nodal values over adjacent cells to increase the degree of continuity of the interpolant, is borrowed from Seibold et al. [115]. However, applying this idea in this context, and doing the averaging in an upwind fashion is novel.

The scheme arrived at in this way is no longer optimally local. However, the sequence of operations described here can be done on an unstructured triangle or tetrahedron mesh. This makes this approach suitable for use with an unstructured mesh that conforms to a complicated boundary. We should mention here that our approach to estimating T_{xy} is referred to as twist estimation in the computer-aided design (CAD) community [50], where other approaches have been proposed [25, 62]. We leave adapting these ideas to the present context for future work.

Marching the amplitude

In this section we show how to compute a numerical approximation of α, denoted $A : \Omega_h \to \mathbb{R}$. One simple approach would be to discretize (3.4) using upwind finite differences and compute $A(\hat{\boldsymbol{x}})$ using `valid` nodes after $T(\hat{\boldsymbol{x}})$ and $\nabla T(\hat{\boldsymbol{x}})$. One potential shortcoming of this approach is that A is singular at caustics. Instead, we will explore using paraxial raytracing to compute A in this section [98]. The background

material on paraxial raytracing used in this section can be found in more detail in M. Popov's book [97].

The basic idea of paraxial raytracing is to consider a fixed, central ray, which we denote $\boldsymbol{\varphi}_0$, and a surrounding tube of rays, parametrized by:

$$\boldsymbol{\varphi}(\sigma, \boldsymbol{q}) = \boldsymbol{\varphi}_0(\sigma) + \boldsymbol{E}(\sigma)\boldsymbol{q}, \tag{3.41}$$

where $\boldsymbol{E} : [0, L] \to \mathbb{R}^{n\times(n-1)}$ is an orthogonal matrix such that $\boldsymbol{E}^\top \boldsymbol{\varphi}_0' \equiv 0$. For each $\boldsymbol{q}$, the corresponding ray should satisfy the Euler-Lagrange equations for (1.4). If we let $c_0(\sigma) = c(\boldsymbol{\varphi}_0(\sigma))$, where $c = 1/s$, then $\boldsymbol{q}$ along with the conjugate momenta $\boldsymbol{p}$ (the exact form of which is not important in this instance) will satisfy:

$$\begin{bmatrix} d\boldsymbol{q}/d\sigma \\ d\boldsymbol{p}/d\sigma \end{bmatrix} = \begin{bmatrix} & c_0(\sigma)\boldsymbol{I} \\ \frac{-1}{c_0(\sigma)^2}\ \frac{\partial^2 c}{\partial \boldsymbol{q}\partial \boldsymbol{q}^\top}\Big|_{\boldsymbol{q}=0} & \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}. \tag{3.42}$$

If we let $\boldsymbol{Q}(\sigma), \boldsymbol{P}(\sigma) : [0, L] \to \mathbb{R}^{(n-1)\times(n-1)}$ be a linearly independent set of solutions to (3.42), then along the central ray, the amplitude satisfies:

$$A(\boldsymbol{\varphi}_0(\sigma)) = \sqrt{\frac{c_0(\sigma)}{|\det(\boldsymbol{Q}(\sigma))|}} A(\boldsymbol{\varphi}_0(0)). \tag{3.43}$$

Note that when we compute an update, we obtain a cubic path $\boldsymbol{\varphi}$ approximating a ray of (1.4), such that $\boldsymbol{\varphi}(0) = \boldsymbol{x}_{\boldsymbol{\lambda}}$ and $\boldsymbol{\varphi}(L) = \hat{\boldsymbol{x}}$.

The quantity $|\det(\boldsymbol{Q}(\sigma))|$ is known as the geometric spreading along the ray tube. We denote it $J(\sigma)$. Letting A denote a polynomial approximation of A off of the grid nodes in Ω_h, using (3.43), we can compute $A(\hat{\boldsymbol{x}})$ from:

$$A(\hat{\boldsymbol{x}}) = \sqrt{\frac{c_0(L_{\boldsymbol{\lambda}})}{|\det(\boldsymbol{Q}(L_{\boldsymbol{\lambda}}))|}} \mathsf{A}(\boldsymbol{x}_{\boldsymbol{\lambda}}) = \sqrt{\frac{c(\hat{\boldsymbol{x}})}{J(\boldsymbol{x}_{\boldsymbol{\lambda}})}} \mathsf{A}(\boldsymbol{x}_{\boldsymbol{\lambda}}). \tag{3.44}$$

Since this depends on $\boldsymbol{Q}(L)$, we must solve (3.42) along $\boldsymbol{\varphi}$, requiring us to provide initial conditions at $\sigma = 0$. Note that if we set $\sigma = 0$ in (3.43), we can see that $|\det(\boldsymbol{Q}(0))| = c_0(0)$ is necessary. A simple choice for the initial conditions for $\boldsymbol{Q}$ is

$\boldsymbol{Q}(0) = c_0(0)^{1/n}\boldsymbol{I}$. This assumes that we aren't too close to a point source, where A is singular.

To find initial conditions for $\boldsymbol{P}$, first expand τ in a Taylor series orthogonal to the central ray, i.e. in the coordinates $\boldsymbol{q}$. Doing this, we find that:

$$\tau(\varphi(\sigma, \boldsymbol{q})) = \tau(\varphi_0(\sigma)) + \frac{1}{2}\boldsymbol{q}^\top \left.\frac{\partial^2 \tau}{\partial \boldsymbol{q} \partial \boldsymbol{q}^\top}\right|_{\boldsymbol{q}=0} \boldsymbol{q} + O(\boldsymbol{q}^3). \tag{3.45}$$

In this Taylor expansion, the linear term disappears since the rays and wavefronts are orthogonal. If we let:

$$\boldsymbol{\Gamma} = \left.\frac{\partial^2 \tau}{\partial \boldsymbol{q} \partial \boldsymbol{q}^\top}\right|_{\boldsymbol{q}=0}, \tag{3.46}$$

we find that $\boldsymbol{\Gamma}$ satisfies the matrix Riccati equation:

$$\frac{d\boldsymbol{\Gamma}}{d\sigma} + c_0\boldsymbol{\Gamma}^2 + \frac{1}{c_0^2}\left.\frac{\partial^2 c}{\partial \boldsymbol{q} \partial \boldsymbol{q}^\top}\right|_{\boldsymbol{q}=0} = 0. \tag{3.47}$$

The standard way to solve (3.47) is to use the ansatz $\boldsymbol{\Gamma} = \boldsymbol{P}\boldsymbol{Q}^{-1}$, which, indeed, leads us back to (3.42). However, this viewpoint furnishes us with the initial conditions for $\boldsymbol{P}$, since $\boldsymbol{\Gamma}(0)$ can now be readily computed from $\nabla^2 \mathsf{T}(\boldsymbol{x}_{\boldsymbol{\lambda}})$.

Marching the amplitude of a linear speed of sound As a simple but important test case, we consider a problem with a constant speed of sound, i.e.:

$$s(\boldsymbol{x}) = \frac{1}{c(\boldsymbol{x})}, \qquad c(\boldsymbol{x}) = v_0 + \boldsymbol{v}^\top \boldsymbol{x}. \tag{3.48}$$

In this case, (3.42) simplifies considerably since $\nabla^2 c \equiv 0$, implying $\boldsymbol{P}(\sigma) \equiv \boldsymbol{P}(0) = \boldsymbol{\Gamma}(0)\boldsymbol{Q}(0)$. From this, we can integrate $d\boldsymbol{Q}/d\sigma$ from 0 to L to obtain:

$$\boldsymbol{Q}(L) = \left[\boldsymbol{I} + \left(\int_0^L c(\boldsymbol{\varphi}(\sigma))d\sigma\right)\boldsymbol{\Gamma}(0)\right]\boldsymbol{Q}(0). \tag{3.49}$$

Denote the integral in this expression for $\boldsymbol{Q}(L)$ by ϵ. To evaluate ϵ approximately, we can apply the trapezoid rule to get:

$$\epsilon = \int_0^L c(\boldsymbol{\varphi}(\sigma))d\sigma = L \cdot \left(v_0 + \boldsymbol{v}^\top(\hat{\boldsymbol{x}} + \boldsymbol{x}_{\boldsymbol{\lambda}})/2\right) + O(L^2), \tag{3.50}$$

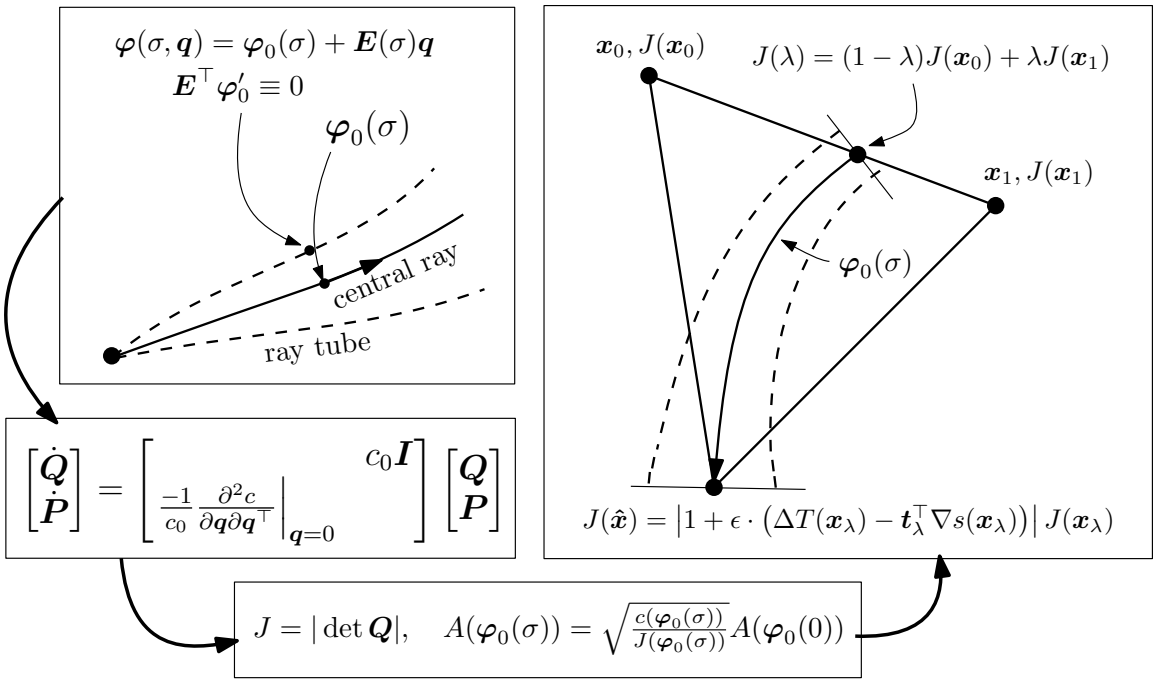


Figure 3.5: An overview of our approach to marching J in a semi-Lagrangian fashion. By solving (3.9), we parametrize the update ray $\boldsymbol{\varphi}$. We think of this as a fixed central ray, $\boldsymbol{\varphi}_0$. We integrate the equations of motion for a small ray tube surrounding φ_0, which allows us to compute propagate a linear approximation of the geometric spreading at $\boldsymbol{x}_\lambda$ to $\hat{\boldsymbol{x}}$.

which implies that $|\epsilon| = O(L)$. The fact that the error is $O(L^2)$ in this case follows from usual error bound for the trapezoid rule and the fact that $\max_{0 \le \sigma \le L} |\boldsymbol{\varphi}''(\sigma)| = O(L^{-1})$, by our choice of parametrization.

We would like to develop a simple update rule for the geometric spreading. First, note that the determinant satisfies the following identity:

$$\det(\boldsymbol{I} + \epsilon \boldsymbol{\Gamma}(0)) = 1 + \epsilon \operatorname{tr}(\boldsymbol{\Gamma}(0)) + O(\epsilon^2). \tag{3.51}$$

Next, recall that $\boldsymbol{E}(0)$ is an orthogonal matrix such that $\boldsymbol{E}(0)^\top \boldsymbol{\varphi}_0' = \boldsymbol{E}(0)^\top \boldsymbol{t}_{\boldsymbol{\lambda}} = 0$. Let $\boldsymbol{U} = \begin{bmatrix} \boldsymbol{E}(0) & \boldsymbol{t}_{\boldsymbol{\lambda}} \end{bmatrix}$ and write:

$$\operatorname{tr} \nabla^2 T(\boldsymbol{x}_{\boldsymbol{\lambda}}) = \operatorname{tr} \boldsymbol{U}^\top \nabla^2 T(\boldsymbol{x}_{\boldsymbol{\lambda}}) \boldsymbol{U} = \operatorname{tr} \boldsymbol{E}(0)^\top \nabla^2 T(\boldsymbol{x}_{\boldsymbol{\lambda}}) \boldsymbol{E}(0) + \boldsymbol{t}_{\boldsymbol{\lambda}}^\top \nabla^2 T(\boldsymbol{x}_{\boldsymbol{\lambda}}) \boldsymbol{t}_{\boldsymbol{\lambda}}. \tag{3.52}$$

By definition, $\boldsymbol{\Gamma}(0) = \boldsymbol{E}(0)^\top \nabla^2 T(\boldsymbol{x}_{\boldsymbol{\lambda}}) \boldsymbol{E}(0)$. Taking the gradient of (1.4), we get:

$$\nabla^2 T(\boldsymbol{x}_{\boldsymbol{\lambda}}) \nabla T(\boldsymbol{x}_{\boldsymbol{\lambda}}) = s(\boldsymbol{x}_{\boldsymbol{\lambda}}) \nabla s(\boldsymbol{x}_{\boldsymbol{\lambda}}), \tag{3.53}$$

which leads immediately to:

$$\boldsymbol{t}_{\boldsymbol{\lambda}}^{\top}\nabla^2 T(\boldsymbol{x}_{\boldsymbol{\lambda}})\boldsymbol{t}_{\boldsymbol{\lambda}} = \boldsymbol{t}_{\boldsymbol{\lambda}}^{\top}\nabla s(\boldsymbol{x}_{\boldsymbol{\lambda}}), \tag{3.54}$$

noting that $\boldsymbol{t}_{\boldsymbol{\lambda}} = \nabla T(\boldsymbol{x}_{\boldsymbol{\lambda}})/\|\nabla T(\boldsymbol{x}_{\boldsymbol{\lambda}})\|$. Combining (3.52) and (3.54) gives:

$$\operatorname{tr}\boldsymbol{\Gamma}(0) = \Delta T(\boldsymbol{x}_{\boldsymbol{\lambda}}) - \boldsymbol{t}_{\boldsymbol{\lambda}}^{\top}\nabla s(\boldsymbol{x}_{\boldsymbol{\lambda}}), \tag{3.55}$$

since $\operatorname{tr}\nabla^2 T(\boldsymbol{x}_{\boldsymbol{\lambda}}) = \Delta T(\boldsymbol{x}_{\boldsymbol{\lambda}})$. This gives the following update for J:

$$J(\hat{\boldsymbol{x}}) = \left|1 + \epsilon \cdot \left(\Delta T(\boldsymbol{x}_{\boldsymbol{\lambda}}) - \boldsymbol{t}_{\boldsymbol{\lambda}}^{\top}\nabla s(\boldsymbol{x}_{\boldsymbol{\lambda}})\right)\right| \cdot \mathsf{J}(\boldsymbol{x}_{\boldsymbol{\lambda}}). \tag{3.56}$$

Here, J denotes a local polynomial approximation to J. This can be computed directly from data immediately available after solving the optimization problem that determines $T(\hat{\boldsymbol{x}})$ and $\nabla T(\hat{\boldsymbol{x}})$. The outline of our approach to computing $J(\hat{\boldsymbol{x}})$ is depicted in Figure 3.5.

Initial data for J and A Determing the initial data for the amplitude is involved [8, 9, 97, 107], and detailed consideration of this problem is outside the scope of this work. Instead, we note that for a point source in 2D, the following hold approximately near the point source:

$$J(\boldsymbol{x}) \sim |\boldsymbol{x}|, \qquad A(\boldsymbol{x}) \sim \frac{e^{i\pi/4}}{2\sqrt{2\pi\omega}}\sqrt{\frac{c(\boldsymbol{x})}{J(\boldsymbol{x})}}. \tag{3.57}$$

See Popov for quick derivations of these approximations [97]. In our test problems, we initialize J to $|\boldsymbol{x}|$ near the point source, march J according to (3.56) where $\mathsf{J} = (1-\lambda)J(\boldsymbol{x}_1) + \lambda J(\boldsymbol{x}_2)$, and compute the final amplitude from:

$$A(\boldsymbol{x}) = \frac{e^{i\pi/4}}{2\sqrt{2\pi\omega}}\sqrt{\frac{c(\boldsymbol{x})}{J(\boldsymbol{x})}}. \tag{3.58}$$

We emphasize that this is only valid for two-dimensional problems. The same sort of approach can be used for 3D problems, but (3.57) must be modified.

Marching the amplitude for more general slowness functions The update given by (3.56) is valid if we approximate the speed function $c = 1/s$ with a piecewise linear function with nodal values taken from $c(\boldsymbol{x})$, where $\boldsymbol{x} \in \Omega_h$. This should be a reasonable thing to do, since the update rule given by (3.56) in this case appears to be $O(h^2)$ accurate. Since the accuracy of $\nabla^2 T$ computed by our method is limited, we should not expect to be able to obtain much better than $O(h)$ accuracy for J. That said, a more accurate update for J could be obtained by numerically integrating (3.42).

3.8 Numerical experiments

In this section, we first present a variety of test problems which differ primarily in the choice of slowness function s. The choices of s range from simple, such as $s \equiv 1$ (an overly simplified but reasonable choice for speed of sound in room acoustics), to more strongly varying. We then present experimental results for our different JMMs as applied to these different slowness functions, demonstrating the significant effect the choice of s has on solver accuracy. The solvers used in these experiments are:

- `JMM1`: $\boldsymbol{\varphi}$ is approximated using a cubic curve, and tangent vectors are found by solving 3.23.
- `JMM2`: $\boldsymbol{\varphi}$ is approximated using a cubic curve, with $\hat{\boldsymbol{t}}$ optimized from 3.28 and $\boldsymbol{t}_{\boldsymbol{\lambda}}$ found from Hermite interpolation at the base of the update.
- `JMM3`: $\boldsymbol{\varphi}$ is approximated using a quadratic curve, with its tangent vectors being found by optimizing.
- `JMM4`: `JMM2` combined with the cell-marching method described in section 3.7.

We also plot the same results obtained by the FMM [116] and `olim8_mp0` [101]. We do not include least squares fits for these solvers in our tables. They are mostly $O(h)$,

with some exceptions for ∇T as computed by the FMM; some light is shed on this in section 3.9.

We note that s does not significantly affect the runtime of any of our solvers—formally, our solvers run in $O(|\Omega_h| \log |\Omega_h|)$ time, where the constant factors are essentially insensitive to the choice of s. We note that the cost of updating the heap is very small compared to the cost of doing updates. Since only $|\Omega_h|$ updates must be computed, the CPU time of the solver effectively scales like $O(|\Omega_h|)$ for all problem sizes considered in this paper.

Test problems

In this section, we provide details for the test problems used in our numerical tests.

Constant slowness with a point source For this problem, the slowness and solution are given by:

$$s \equiv 1, \qquad \tau(\boldsymbol{x}) = \|\boldsymbol{x}\|. \tag{3.59}$$

We take the domain to be $\Omega = [-1, 1] \times [-1, 1] \subseteq \mathbb{R}^2$. To control the size of the discretized domain, we let $M > 0$ be an integer and set $h = 1/M$, from which we define Ω_h accordingly. We place a point source at $\boldsymbol{x}^\circ = (0, 0) \in \Omega_h$. The set of initial boundary data locations given by is $\Gamma_h = \{\boldsymbol{x}^\circ\}$, with boundary conditions given by $g(\boldsymbol{x}^\circ) = 0$.

Linear speed with a point source (#1) Our next test problem has a linear velocity profile. This might model the variation in the speed of sound due to a linear temperature gradient (e.g., in a large room). The slowness is given by [52, 125]:

$$s(\boldsymbol{x}) = \left[\frac{1}{s_0} + \boldsymbol{v}^\top \boldsymbol{x}\right]^{-1}, \tag{3.60}$$

where $s_0 > 0$, and $\boldsymbol{v} \in \mathbb{R}^2$ are parameters. The solution is given by:

$$\tau(\boldsymbol{x}) = \frac{1}{\|\boldsymbol{v}\|} \cosh^{-1}\left(1 + \frac{1}{2} s_0 s(\boldsymbol{x}) \|\boldsymbol{v}\|^2 \|\boldsymbol{x}\|^2\right). \tag{3.61}$$

For our first test with a linear speed function, we take $s_0 = 1$ and $\boldsymbol{v} = (0.133, -0.0933)$. For this problem, $\Omega = [-1, 1] \times [-1, 1]$, $\Gamma_h = \{\boldsymbol{x}^\circ\}$, and $g(\boldsymbol{x}^\circ) = 0$.

Linear speed with a point source (#2) For our second linear speed test problem, we set $s_0 = 2$ and $\boldsymbol{v} = (0.5, 0)$ as in [106]. For this problem, we let $\Omega = [0, 1] \times [0, 1]$, discretize into M nodes along each axis, and define Ω_h accordingly (i.e., $|\Omega_h| = M^2$, with $M = h^{-1}$). We take $\boldsymbol{x}^\circ$, Γ_h, and g to be same as in the previous two test problems.

A nonlinear slowness function involving a sine function For $\boldsymbol{x} = (x_1, x_2)$, we set:

$$\tau(\boldsymbol{x}) = \frac{x_1^2}{2} + 2\sin\left(\frac{x_1 + x_2}{2}\right)^2. \tag{3.62}$$

This eikonal has a unique minimum, $\tau(0, 0) = 0$, and is strictly convex in $\Omega = [-1, 1] \times [-1, 1]$. This lets us determine the slowness from the eikonal equation, giving:

$$s(\boldsymbol{x}) = \sqrt{\sin(x_1 + x_2)^2 + \big(x_1 + \sin(x_1 + x_2)\big)^2}. \tag{3.63}$$

For this test problem, we take Γ_h and Ω_h as in the constant slowness point source problem.

Sloth A slowness function called "sloth" (jargon from geophysics) is taken from Example 1 of Fomel et al. [52]:

$$s(\boldsymbol{x}) = \sqrt{s_0^2 + 2\boldsymbol{v}^\top \boldsymbol{x}}. \tag{3.64}$$

For our test with this slowness function, we set $s_0 = 2$, and $\boldsymbol{v} = (0, -3)$. In this case, to avoid shadow zones formed by caustics, we take $\Omega = [0, \frac{1}{2}] \times [0, \frac{1}{2}]$. The discretized domain and boundary data are determined analogously to the earlier cases.

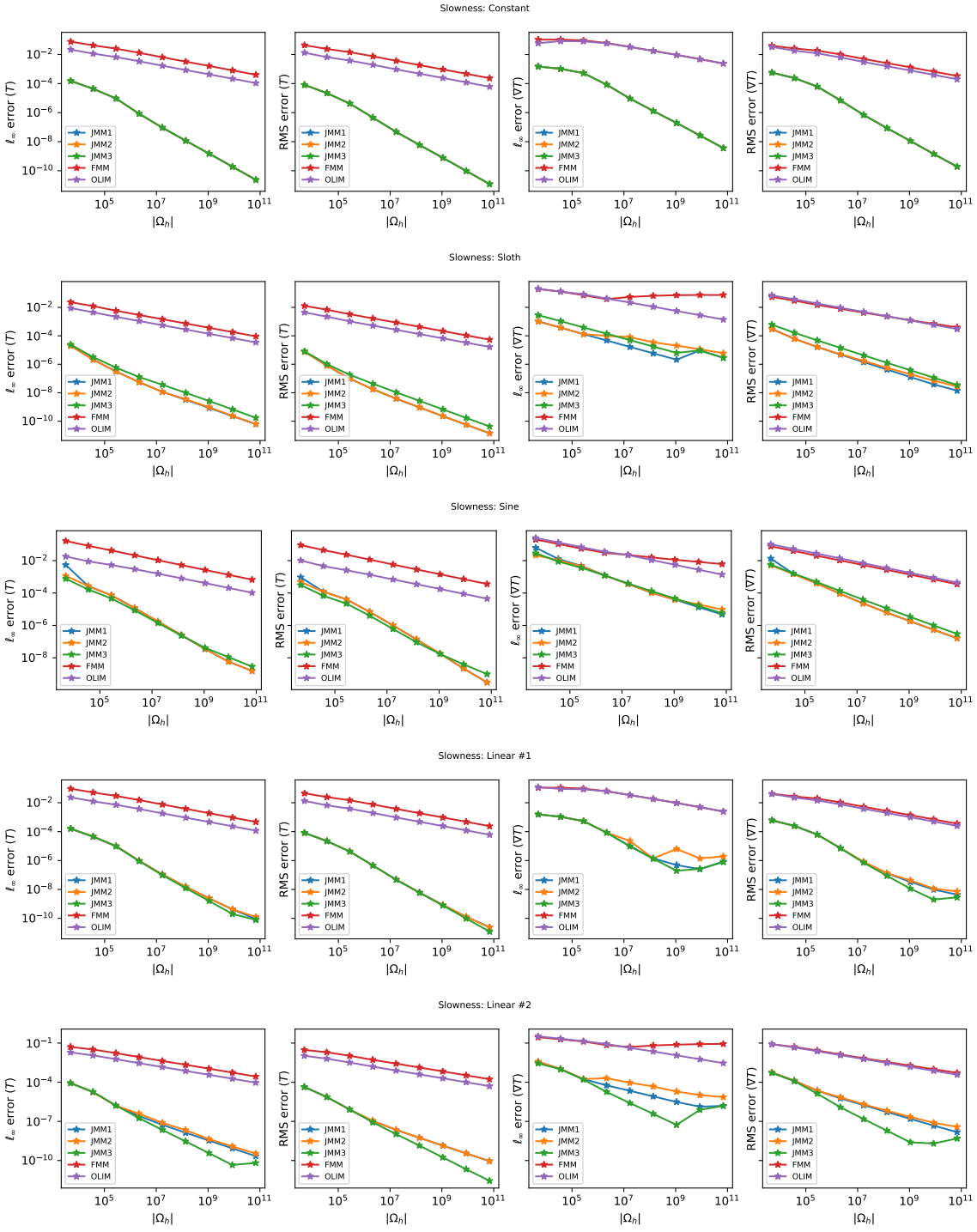


Figure 3.6: Plots comparing domain size ($|\Omega_h|$) and ℓ_∞ and RMS errors for T and ∇T.

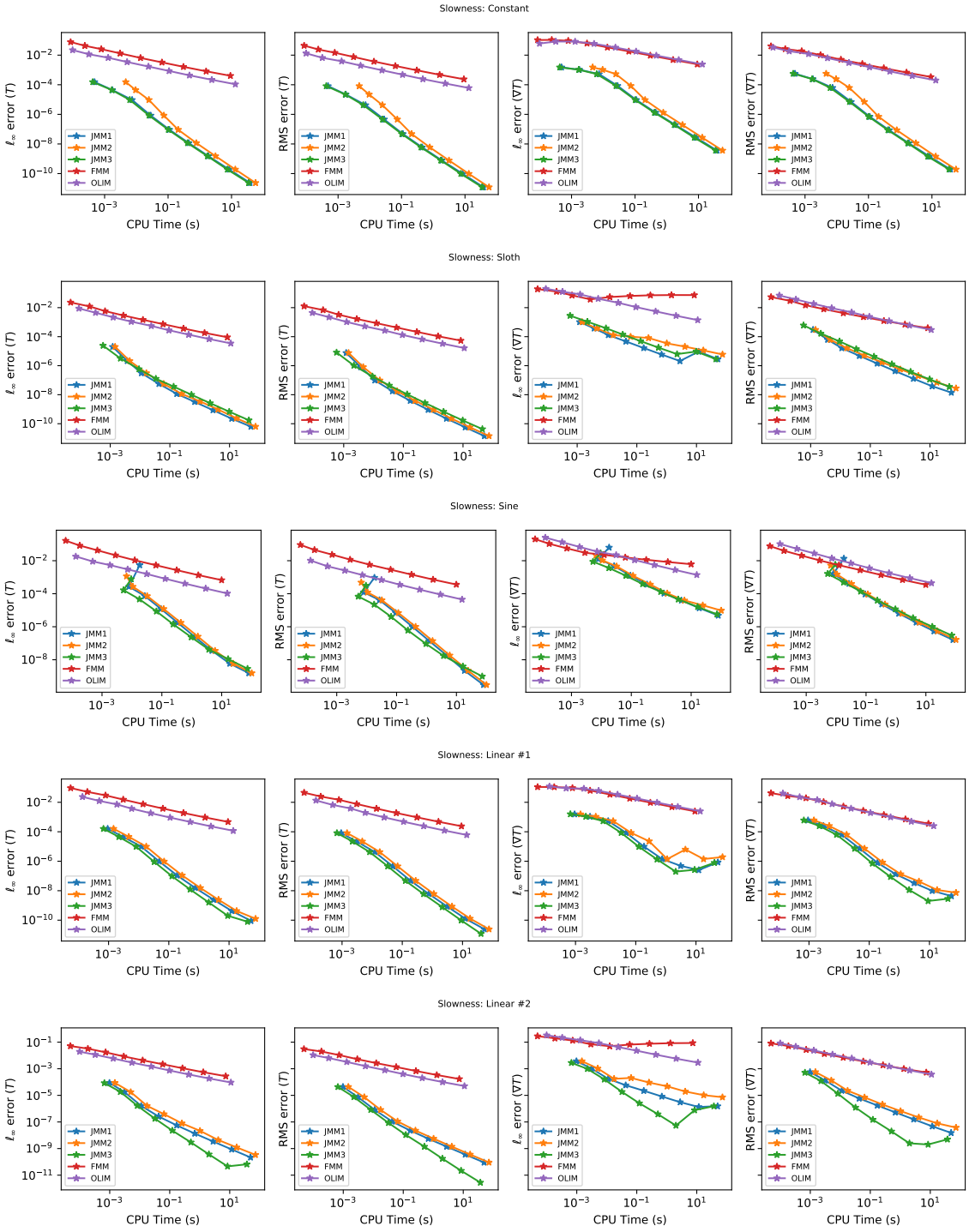


Figure 3.7: Plots comparing CPU runtime in seconds and errors.

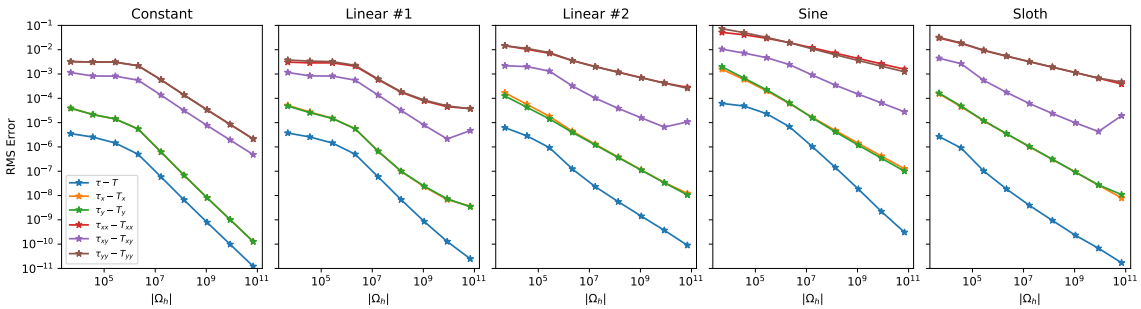


Figure 3.8: Domain size vs. RMS error for `JMM4`.

	JMM	$E_{\max}(T)$	$E_{\text{RMS}}(T)$	$E_{\max}(\nabla T)$	$E_{\text{RMS}}(\nabla T)$
Constant	#1	2.87	2.87	2.28	2.72
	#2	2.87	2.87	2.28	2.72
	#3	2.87	2.87	2.28	2.72
Linear #1	#1	2.77	2.85	2.14	2.52
	#2	2.77	2.85	1.70	2.48
	#3	2.86	2.87	2.28	2.73
Linear #2	#1	2.48	2.52	1.70	1.97
	#2	2.38	2.52	1.16	1.88
	#3	3.03	3.03	2.70	3.02
Sine	#1	2.76	2.57	1.77	2.09
	#2	2.51	2.46	1.58	1.94
	#3	2.37	2.38	1.54	1.79
Sloth	#1	2.39	2.48	1.49	1.84
	#2	2.37	2.47	0.87	1.73
	#3	2.15	2.21	1.47	1.76

Table 3.1: The order of convergence p for each combination of test problems and solvers, computed for different types of errors and fit as Ch^p.

	$\tau - T$	$\tau_x - T_x$	$\tau_y - T_y$	$\tau_{xx} - T_{xx}$	$\tau_{xy} - T_{xy}$	$\tau_{yy} - T_{yy}$
Constant	3.09	3.11	3.11	2.01	2.05	2.01
Linear #1	2.99	2.43	2.40	1.39	2.01	1.39
Linear #2	2.10	1.76	1.72	0.77	1.25	0.77
Sine	2.91	1.80	1.89	0.73	1.31	0.80
Sloth	2.03	1.76	1.75	0.75	1.33	0.76

Table 3.2: The order of convergence p for `JMM4` for each component of the total 2-jet of τ, computed from least squares fits of the RMS error. The fits only incorporate the 4th through the 8th problem sizes to avoid artifacts for small and large problem sizes. See Figure 3.8.

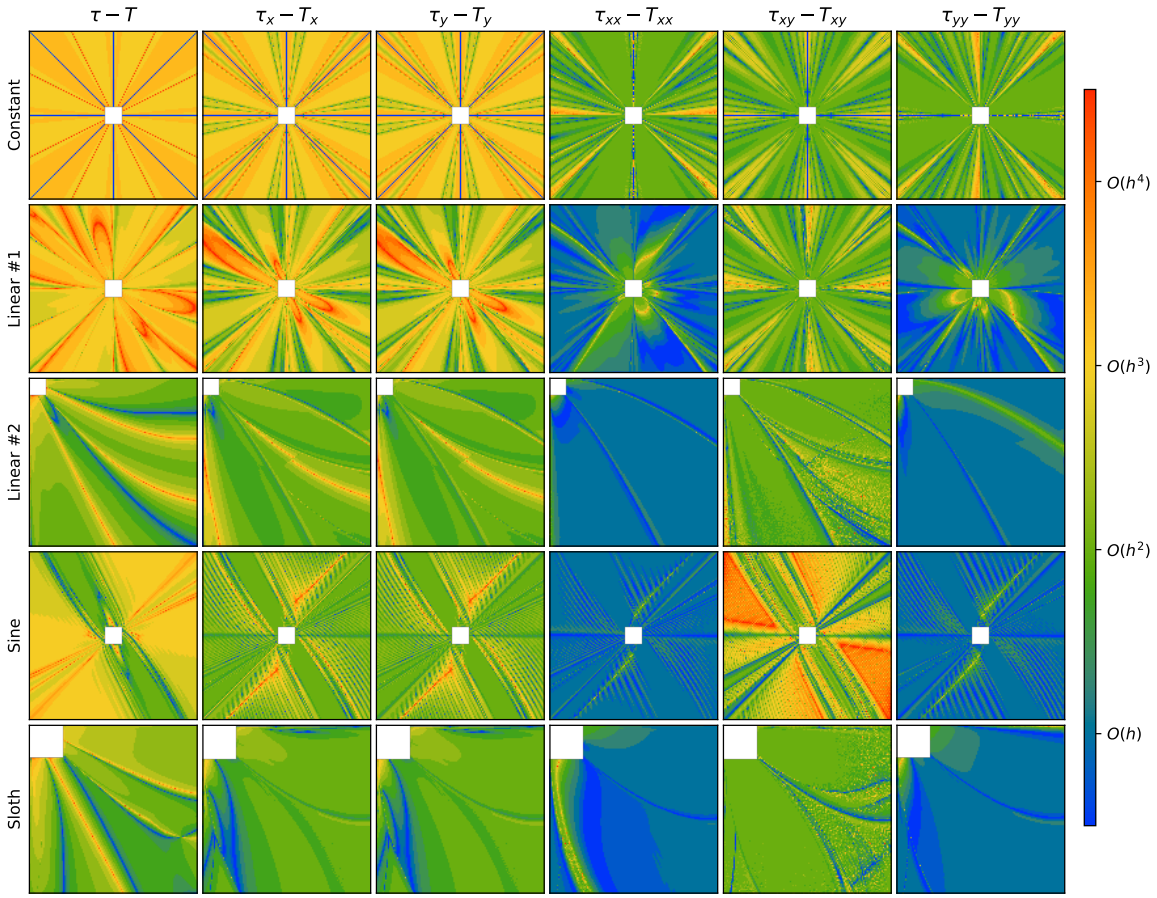


Figure 3.9: A plot of the pointwise convergence at each point in Ω_h for `JMM4`. To obtain these plots, starting with $N = 129$, we decimate each larger problem size (up to $N = 2,049$) to a 129×129 grid, and do a least squares fit at each point. This gives us an estimate of the order of convergence at each point.

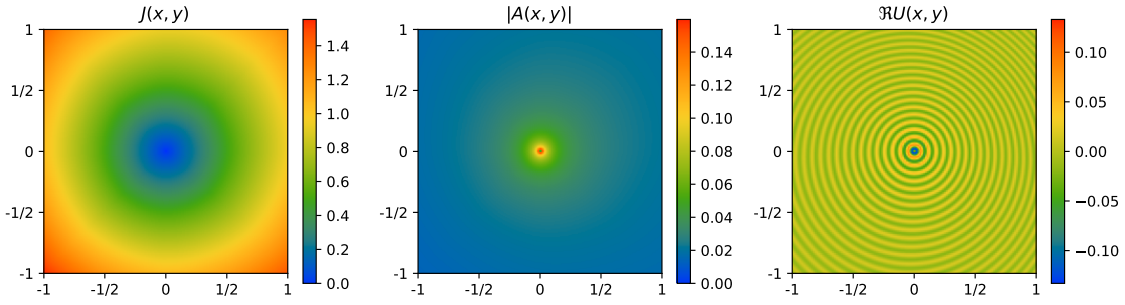


Figure 3.10: Plots related to computing the amplitude and a numerical approximation to the solution to $(\Delta + \omega^2 s(\boldsymbol{x})^2)u(\boldsymbol{x}) = \delta(\boldsymbol{x})$, denoted $U(\boldsymbol{x})$ for the Linear #1 test problem. *Left*: the geometric spreading. *Middle*: the amplitude function. *Right*: the numerical solution U.

Experimental results

The results of our numerical experiments evaluating the JMMs described in Section 3.4 are presented in Table 3.1 and Figures 3.6 and 3.7. The numerical tests for `JMM4`, which uses cell marching, are given in Table 3.2 and Figures 3.8 and 3.9. An example where the geometric spreading and amplitude are computed using cell marching method is shown in Figure 3.10.

For more benign choices of s, the errors generally convergence with $O(h^3)$ accuracy for T and $O(h^2)$ accuracy for ∇T in the RMS error. For the special case of $s \equiv 1$, the gradients also converge with nearly $O(h^3)$ accuracy. For more challenging nonlinear choices of s, the eikonal converges with somewhere between $O(h^2)$ and $O(h^3)$ accuracy, while the gradient converges with nearly $O(h^2)$ accuracy.

We note that in some cases the gradient begins to diverge for large problem sizes. This occurs because our tolerance for minimizing F is not small enough, and also because $\nabla^2 F$ is $O(h)$. For our application, our goal is to save memory and compute time by using a higher-order solver; it is unlikely we would solve problems with such a fine discretization in practice. At the same time, choosing the tolerance for numerical minimization based on h is of interest—partly to see how much time can be saved for coarser problems, but also to determine to what extent the full order of convergence can be maintained using different floating point precisions.

The JMMs using cubic approximations for $\boldsymbol{\varphi}$ tend to perform better than those using quadratic approximations when s is nonlinear and the characteristics of (1.4) are not circular arcs. When s corresponds to a linear speed of sound, the JMMs with quadratic $\boldsymbol{\varphi}$ are a suitable choice, generally outperforming the "cubic $\boldsymbol{\varphi}$" solvers, exhibiting cubically (or nearly cubically) convergent RMS errors in T and ∇T. This is a useful finding since the simplified solver requires fewer floating-point operations per update, and since linear speed of sound profiles (e.g., as a function of a linear temperature profile) are a frequently occurring phenomenon in room acoustics.

3.9 Theoretical results

In this section we present two theoretical results of interest:

- We prove consistency for the JMM presented in 3.4. This results suggests that $O(h^2)$-accurate values of T and $O(h)$ accurate ∇T values are to be expected in general, with certain choices of s giving rise to $O(h^3)$ values of T and $O(h^2)$, or even $O(h^3)$, values of ∇T.
- We demonstrate a counterexample that shows how using a 4-point stencil can degrade the overall order of convergence of a jet marching method, and how the use of an 8-point stencil overcomes this problem.

Consistency and convergence of the quadratic curve JMM

As we have seen, our numerical results indicate that we can expect between $O(h^2)$ and $O(h^3)$ accuracy for $T(\boldsymbol{x})$ and roughly $O(h^2)$ accuracy for $\nabla T(\boldsymbol{x})$. Our goal is to determine the conditions under which $O(h^3)$-accurate T and $O(h^2)$-accurate ∇T obtains. Establishing theoretical lower bounds on the order of convergence for T and ∇T requires tedious calculations and nuanced considerations, making the numerical analysis of this solver an interesting and challenging problem in its own right. We defer an extensive study of this problem to future work. Here, we simply present consistency results for `JMM3` operating on $\Omega \subseteq \mathbb{R}^2$ and preview our convergence results.

Theorem 6. *Let $\Omega \subseteq \mathbb{R}^2$, assume that $s \in C^2(\Omega)$, and let $(\hat{\boldsymbol{x}}, \boldsymbol{x}_1, \boldsymbol{x}_2)$ be an update triangle where $\texttt{state}(\hat{\boldsymbol{x}}) = \texttt{trial}$, $\texttt{state}(\boldsymbol{x}_1) = \texttt{valid}$, and $\texttt{state}(\boldsymbol{x}_2) = \texttt{valid}$. Assume that no caustic is incident on the update triangle, and that the ray arriving at $\hat{\boldsymbol{x}}$ crosses the segment $[\boldsymbol{x}_1, \boldsymbol{x}_2]$. Then:*

$$|\tau(\hat{\boldsymbol{x}}) - T(\hat{\boldsymbol{x}})| = O(h^4), \qquad \|\nabla\tau(\hat{\boldsymbol{x}}) - \nabla T(\hat{\boldsymbol{x}})\| = O(h^2). \tag{3.65}$$

Proof. A detailed proof this theorem requires lengthy but straightforward calculations, and will be presented elsewhere; so, instead, we give an outline of the proof.

Let $\boldsymbol{x}_\lambda = (1-\lambda)\boldsymbol{x}_1 + \lambda \boldsymbol{x}_2$, we define:

$$f(\lambda) = \tau(\boldsymbol{x}_\lambda) + \int_0^{L_\lambda} s(\boldsymbol{\varphi}(\sigma))\|\boldsymbol{\varphi}'(\sigma)\| d\sigma, \tag{3.66}$$

where $L_\lambda = \|\hat{\boldsymbol{x}} - \boldsymbol{x}_\lambda\|$, and let $F(\boldsymbol{x}_\lambda, \boldsymbol{t})$ be the cost functional given by (3.18) or (3.22). Note that:

$$\tau(\hat{\boldsymbol{x}}) = \min_{0\leq\lambda\leq 1} f(\lambda), \qquad T(\hat{\boldsymbol{x}}) = \min_{0\leq\lambda\leq 1} F(\boldsymbol{x}_\lambda, \boldsymbol{t}(\boldsymbol{x}_\lambda)), \tag{3.67}$$

and that the gradients can recovered via:

$$\nabla\tau(\hat{\boldsymbol{x}}) = s(\hat{\boldsymbol{x}})\frac{\boldsymbol{\varphi}'(L_\lambda)}{\|\boldsymbol{\varphi}'(L_\lambda)\|}, \qquad \nabla T(\hat{\boldsymbol{x}}) = s(\hat{\boldsymbol{x}})\boldsymbol{t}(\hat{\boldsymbol{x}}). \tag{3.68}$$

From these expressions, using multivariable calculus and classical interpolation theory [127], we show that:

$$|f(\lambda) - F(\lambda)| \leq Ch^4, \qquad 0 \leq h \leq 1, \tag{3.69}$$

where $C > 0$ is a constant. Then, using this fact, we can show:

$$\left| \min_{0\leq\lambda\leq 1} f(\lambda) - \min_{0\leq\lambda\leq 1} F(\lambda) \right| \leq Ch^4. \tag{3.70}$$

This establishes that $|\tau(\hat{\boldsymbol{x}}) - T(\hat{\boldsymbol{x}})| = O(h^4)$. To bound the error in the gradients, we first show that $|f'(\lambda) - F'(\lambda)| = O(h^3)$. Then, we prove that:

$$F''(\lambda) \geq C'h, \qquad 0 \leq \lambda \leq 1, \tag{3.71}$$

where $C' > 0$ is another positive constant. It follows from the intermediate value theorem that:

$$\left| \arg\min_{0\leq\lambda\leq 1} f(\lambda) - \arg\min_{0\leq\lambda\leq 1} F(\lambda) \right| = O(h^2), \tag{3.72}$$

which we use to establish $\|\nabla\tau(\hat{\boldsymbol{x}}) - \nabla T(\hat{\boldsymbol{x}})\| = O(h^2)$. □

The consistency results stated in Theorem 6 might suggest to the reader that the global error for ∇T is $O(h)$ due to error accumulation. It turns out that this is not the case. We conducted an extensive theoretical study for `JMM3` in $\mathbb{R}^2$. The numerical solution at a particular grid node is determined from the solution at some number of "parent" nodes (at most n of them). Let $\texttt{parent}(\boldsymbol{x})$ be this set of nodes, and let:

$$\texttt{parent}(\{\boldsymbol{x}_i\}_{i\in I}) = \bigcup_{i\in I} \texttt{parent}(\boldsymbol{x}_i), \qquad \texttt{parent}(\{\}) = \{\}. \tag{3.73}$$

We note that if $\boldsymbol{x} \in \Gamma_h$, and $g(\boldsymbol{x})$ is chosen appropriately, then $\texttt{parent}(\boldsymbol{x}) = \{\}$. If we let $\texttt{parent}^{(k)}$ denote the k-fold application of the `parent` operator, then, for each $\boldsymbol{x} \in \Omega_h$, let $k \geq 0$ be the minimum integer such that:

$$\text{parent}^{k+1}(\boldsymbol{x}) = \{\}. \tag{3.74}$$

We call k the generation of an update.

If we let $e^{(k)}$ denote the error in T, where k is the generation of the update, and let $g^{(k)}$ denote the corresponding error in ∇T, then our calculations show that:

$$\begin{bmatrix} e^{(k+1)} \\ g^{(k+1)} \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} e^{(k)} \\ g^{(k)} \end{bmatrix} + \begin{bmatrix} O(h^3) \\ O(h^2) \end{bmatrix}, \tag{3.75}$$

where $\boldsymbol{A} \in \mathbb{R}^{2\times 2}$ is an amplification matrix that we obtain explicitly. Our study of $\boldsymbol{A}$'s properties shows that the errors in T accumulate linearly along special curves—if they exist for a given problem—along which the errors in the eikonal at the parent nodes are equal, and will attentuate otherwise. The errors in the gradient attenuate everywhere, resulting in the global $O(h^2)$ error.

Our findings are consistent with our numerical results: the least squares fits for the errors in the eikonal give orders of convergence between two and three, while the gradient mostly second-order accurate, until roundoff errors come into play. We also conducted a separate study showing that the global error for the case of $s \equiv \text{const}$ in exact arithmetic must be $O(h^3)$ for both T and ∇T. The case where $c = 1/s$ is

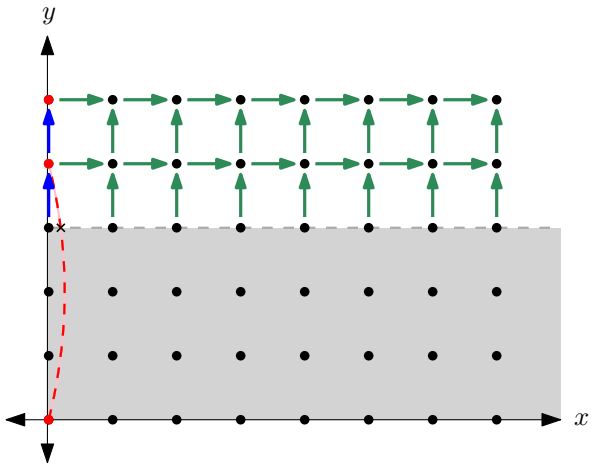


Figure 3.11: A semi-infinite horizontal slab of nodes in Ω_h are initialized with the correct values of τ and $\nabla\tau$. When computing T throughout the rest of the domain using a jet marching method, the first node to be updated will be the lower-left node in the rest of the domain. If a 4-point stencil is used, this results in $O(h^2)$ error which pollutes the rest of the solution.

linear is also a special case of interest to be treated separately, which we have left for future work.

A counterexample demonstrating the need for the 8-point stencil

On a regular grid in $\mathbb{R}^2$, first-order eikonal solvers that use either a 4-point or 8-point stencil are known to work reliably. The use of an 8-point stencil typically improves the error constant by an order of magnitude; in $\mathbb{R}^3$, the differences between the 6-point and 26-point stencils are roughly the same [101]. We are not aware of any thorough studies of the effect of stencils on higher-order solvers.

In this section, we provide a counterexample that demonstrates that using the 4-point stencil can limit the rate of convergence of a JMM to quadratic. Consider the following linear speed function:

$$s : \mathbb{R}^2 \to \mathbb{R}, \qquad \boldsymbol{x} = (x, y) \mapsto \frac{2}{1 + x}. \tag{3.76}$$

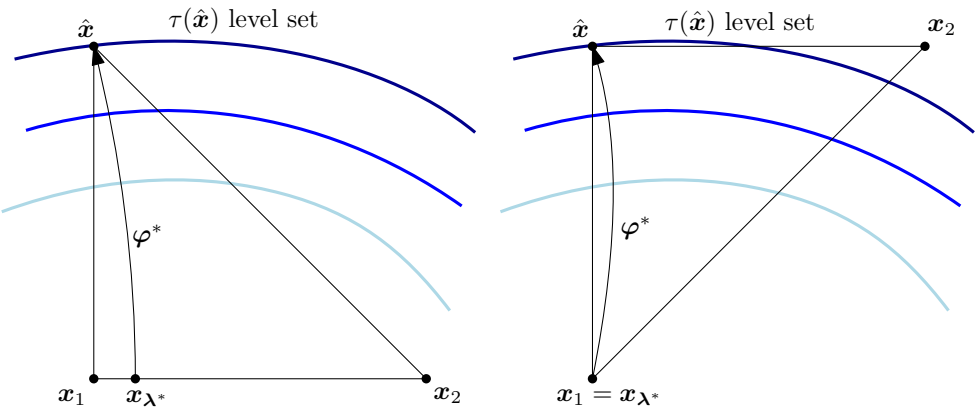


Figure 3.12: What can go wrong with the 4-point stencil. *Left*: the 8-point stencil. In this case, the base of the triangle update is fully upwind of the $\tau(\hat{\boldsymbol{x}})$ level set, which allows $\boldsymbol{x}_{\boldsymbol{\lambda}}$ to be localized with sufficient accuracy. *Right*: the 4-point stencil. One of the nodes of the triangle update is downwind of the $\tau(\hat{\boldsymbol{x}})$ level set. This causes a one-point update to be selected, which incurs $O(h^2)$ error in $T(\hat{\boldsymbol{x}})$, polluting the rest of the solution, thereby degrading the global accuracy of T.

For this choice of s, the same as in test problem "Linear #2", rays arriving at points upper half-plane ($y > 0$) are circular arcs passing through the origin, whose centers are located on the vertical line $x = -1$.

Imagine that we solve (1.4) on a grid:

$$\Omega_h = \left\{(h \cdot i, h \cdot j) : (i, j) \in \{0, 1, \ldots\}^2\right\}. \tag{3.77}$$

where $h > 0$. Let $n_{\texttt{slab}} = \lfloor \frac{C}{h} \rfloor$, where C is a small, positive constant (e.g., $C = 0.1$), and initialize each $\boldsymbol{x}$ in the horizontal slab:

$$\{(h \cdot i, h \cdot j) : (i, j) \in \{0, 1, \ldots\} \times [0, n_{\texttt{slab}}]\} \subseteq \Omega_h, \tag{3.78}$$

with the correct values of $\tau(\boldsymbol{x})$ and $\nabla\tau(\boldsymbol{x})$. We will consider what happens when we run a jet marching method to compute T throughout the rest of Ω_h.

First, note that on each slab, the node $\boldsymbol{x} = (0, h \cdot j)$ for some j has the minimal value of T, provided that C is small enough. This will always be the first node to receive a `valid` state, having its T value fixed. Hence, the error of all other nodes in Ω_h will be affected by the error at the nodes on the x_2-axis. We will consider the

difference in computing these T values when using a 4-point versus an 8-point stencil. The general setup is shown in Figure 3.11.

The exact slowness for our choice of s is given by:

$$\tau(\boldsymbol{x}) = 2\cosh^{-1}\left(1 + \frac{x_1^2 + x_2^2}{2(1+x_1)}\right). \tag{3.79}$$

If we solve $\partial\tau/\partial x_1 = 0$ for x_1, choosing the positive root, we obtain the curve:

$$x_1 = -1 + \sqrt{1 + x_2^2}. \tag{3.80}$$

This is the curve along which τ is minimized for each choice of x_2. If we perform an update to compute a value of T along the x_2 axis, there are two different situations to consider depending on the stencil that we use (see Figure 3.12).

If we use the 4-point stencil, when updating nodes $\hat{\boldsymbol{x}} = (0, h\cdot j)$, a one-point update will be selected since one point in the base of the triangle update will be downwind from the $T(\hat{\boldsymbol{x}})$ level set, forcing $\boldsymbol{x_\lambda} = \boldsymbol{x}_1$. The resulting error in $\boldsymbol{x_\lambda}$ is $O(h^2)$, as the onset of the characteristic arriving at $\hat{\boldsymbol{x}}$ is shifted by $O(h^2)$. This results in $O(h^2)$ error in $T(\hat{\boldsymbol{x}})$. On the other hand, if the 8-point stencil is used, both points are upwind of the $T(\hat{\boldsymbol{x}})$ level set, allowing the base of the updating characteristic, $\boldsymbol{x_\lambda}$, to be localized with $O(h^3)$ accuracy.

As a final observation, we can see that the values of ∇T computed by the FMM and `olim8_mp0` in our numerical results are roughly $O(h)$-accurate in most cases. We can see that in some cases, the gradients computed by the FMM diverge. We posit that this difference may be a result of the same phenonemon demonstrated in this section, noting that the FMM uses a 4-point stencil while `olim8_mp0` uses an 8-point stencil. It has often been observed that first-order direct solvers for the eikonal equation compute $O(h)$ accurate gradients. We believe this counterexample sheds some light on the inconsistency in the observed results.

3.10 Online package

To recreate our results, to experiment with these solvers, and to understand their workings, a package has been made available online on GitHub at:

`https://github.com/sampotter/jmm/tree/jmm-sisc-figures`

Details explaining how to obtain this package and to collect the results are available at this link.

3.11 Conclusion

We have presented a family of semi-Lagrangian label-setting methods (à la the fast marching method) which are high-order and compact, which we refer to as jet marching methods (JMMs). We examine a variety of approaches to formulating one of these solvers, and in 2D, provide extensive numerical results demonstrating the efficacy of these approaches. We show how a form of "adaptive" cell-marching can be done which is compatible with our stencil compactness requirements, although this scheme no longer displays optimal locality. We also prove preliminary convergence guarantees for a particular case in 2D.

Our solvers are motivated by problems involving repeatedly solving the eikonal equation in complicated domains where:

- time and memory savings via the use of high-order solvers,
- high-order local knowledge of characteristic directions,
- and compactness of the solver's "stencil" (the neighborhood over which the semi-Lagrangian updates require information)

is paramount. In particular, our goal is to parametrize the multipath eikonal in a complicated polyhedral domain in a work-efficient manner. This solver is a necessary ingredient for carrying out this task.

Apart from this application, we will be continuing to work along the following directions:

- Application of these solvers to regular grids in 3D, which should be straightforward and yield considerable savings over existing approaches, and extension to unstructured simplex meshes in 2D and 3D. Especially in 3D, this problem is more complicated, requiring the computation of "causal stencils" [72, 119].

- Proofs of convergence for each of our approaches in the n-dimensional setting. This requires an extension of the proofs used for this work, but otherwise the idea is the same.

- A more careful characterization of the conditions under which cubic convergence of T and ∇T is obtained.

Chapter 4

A tetrahedron mesh JMM and multiple arrivals in 3D

In this chapter, our goal is to develop an algorithm which is capable of recursively generating eikonal problems, solving them, patching them together into a tree of multiple arrivals, extracting sequences of arrivals at individual points, and doing what is necessary to compute the amplitude for each branch along the way. Throughout, we will assume that c is constant over Ω. Without loss of generality, we can simplify things even further by assuming that $c \equiv 1$ so that solving the eikonal equation is equivalent to computing the distance transform. Afterwards, we can divide the distance by c to obtain the travel time.

As before, we will denote our problem domain by $\Omega \subseteq \mathbb{R}^3$, and its boundary by $\partial\Omega$. Our focus will be closed and compact domains, but, unlike with methods based on solving the wave or Helmholtz equations, interior, exterior, and partially enclosed domains can all be treated with a similar amount of effort, provided that they contain scattering obstacles of similar complexity. We will assume that $\partial\Omega$ is provided as a polygonal mesh, consisting of large, planar facets. Technically speaking, we assume that $\partial\Omega$ is a piecewise linear complex (PLC), so that methods for Delaunay tetra-

hedralization can be applied [34]. The requirement that the mesh be Delaunay does not appear to be necessary for our purposes, but the fact that a uniform tetrahedron mesh is easily produced by good meshing software is advantageous. We have two reasons for focusing on unstructured tetrahedron meshes:

- It is straightforward to generate a tetrahedron mesh with a prescribed fineness from the boundary-representation of a domain using widely available software.
- An unstructured mesh can conform to an unstructured boundary. If we were to discretize Ω using a regular grid, an obstacle which isn't axis-aligned would result in digitization artifacts along the boundary, complicating matters.[1]

Several things need to be done to lift the requirement that c is constant:

- **Develop local updates that incorporate a varying speed of sound.**[2] This can be done using techniques from Chapter 3 adapted to 3D. This is straightforward, but—again, as in Chapter 3—there will likely be an attendant drop in the order of accuracy in the numerical solution T attributed to a varying speed of sound, which will be further exacerbated by a nonuniform mesh. The methods presented in this chapter are formally $O(h^2)$ accurate. We discuss approaches to marching the eikonal in 3D with an even higher formal order of accuracy in Chapter 5.
- **Initialize the eikonal carefully near caustics.** These occur at point sources and in the shadow immediately past a diffracting edge. Dynamic local factoring has been developed in 2D [106]. In future work, we plan on developing an approach along the lines of dynamic local factoring which works in 3D. Because of the more complicated geometry in 3D, the problem is more involved. For

[1] This can be dealt with using an unstructured solver in the boundary layer, but having one solver for the entire domain keeps thing simple [100].

[2] We thank Manyuan Tao for carrying out the initial development of a 3D JMM for a nonconstant speed of sound on a regular grid in 3D.

now, we focus on developing a solution that is good enough for simple room acoustics simulations.

- **Handle free-space caustics.** Allowing c to vary introduces the possibility of caustics developing in free space. This creates two problems: 1) the GA ansatz breaks down, predicting an infinite amplitude on caustics, and 2) multiple arrivals occur in free space. The first problem can be addressed either using the Gaussian beam method [96, 28, 97], or using GTD [13, 14]. The second problem requires us to be able to parametrize the caustic set and recursively generate a new eikonal problem for the scattered field, analogous to what is done for reflections and diffractions in this chapter.

Addressing the extensions listed above does not change the basic structure of the algorithms for computing multiple arrivals presented in this chapter, so we leave them for future work.

4.1 Computing RIRs

To compute room impulse responses (RIRs), at each point, for a particular point source location $\boldsymbol{x}_{\text{src}}$, one approach would be to compute:

$$\left(\tau^{(k)}(\boldsymbol{x}), \frac{\nabla\tau^{(k)}(\boldsymbol{x})}{\|\nabla\tau^{(k)}(\boldsymbol{x})\|}, \alpha^{(k)}(\boldsymbol{x})\right), \qquad k = 0, 1, \ldots, k_{\max} - 1 \tag{4.1}$$

at a prescribed set of nodes, where the superscript "$^{(k)}$" denotes the kth arrival, and where $k_{\max}$ is the maximum number of arrivals. Once we have several arrivals at each point, we can use standard methods for auralizing spatial audio for rendering directional sound [145].

Another approach would be to compute a parametric RIR [109, 110]. In this case, a field of point-to-point RIRs for a fixed source or listener position are encoded as a set of smoothly varying, perceptually relevant fields. In this case, roughly speaking,

we might compute (4.1) for $k = 0, 1$ (corresponding to the first arrival and a single early arrival), along with an estimate of the decay time and early reverberant energy at each point. The decay time can be estimated using a running least squares fit of the energy of each arrival, and the reverberant energy involves summing the energy of each arrival over a fixed time window following $\tau^{(1)}$. These techniques were developed to precompute fields of RIR parameters while stepping a time-domain wave equation solver. As the solver runs, a nonlinear onset detection filter is used to estimate arrival times, while other quantities are integrated from the solution at each point.

4.2 The multipath eikonal

In this section, we discuss how to compute the multipath eikonal for a point source $\boldsymbol{x}_{\text{src}} \in \Omega$. We will initially solve the eikonal equation for a single point source:

$$\begin{cases} \|\nabla\tau(\boldsymbol{x})\| = 1/c, & \text{for } \boldsymbol{x} \in \Omega, \\ \tau(\boldsymbol{x}_{\text{src}}) = 0. \end{cases} \tag{4.2}$$

Solving eq. (4.2) results in the first arrival time throughout Ω, which includes points which were reached by diffracted rays. Since we assume that $\partial\Omega$ is a PLC, we can decompose $\partial\Omega$ into its different connected facets and edges. We will refer to the facets as reflectors, and we will refer to each edge which has an interior angle greater than π as a diffractor. Diffractors are so distinguished because they are capable of generating a shadow.

After solving eq. (4.2), we will iterate over each reflector and diffractor and generate new eikonal problems to solve. Let τ_{in} be the eikonal of the incident field. For a reflector or diffractor S, we will restrict τ_{in} to a subset $\Gamma \subseteq S$, and solve:

$$\begin{cases} \|\nabla\tau_{\text{out}}\| = 1/c, & \text{for } \boldsymbol{x} \in \Omega, \\ \tau_{\text{out}}(\boldsymbol{x}) = \tau_{\text{in}}(\boldsymbol{x}), & \text{for } \boldsymbol{x} \in \Gamma. \end{cases} \tag{4.3}$$

We will discuss how the set Γ consisting of nodes with BCs is chosen (roughly, it will consist of the facets in S with nodes that have a large enough amplitude), and how to set up and solve (4.3) for reflections and diffractions in Section 4.7.

Conditions for a ray to be physical. In Section 1.6, we derived general BCs for the reflected and diffracted eikonal: see (1.47) and (1.48). Since each individual reflector or diffractor is flat, $\partial \boldsymbol{X}/\partial \boldsymbol{\sigma}$ is constant over each scattering feature. If $\boldsymbol{\nu}_r$ is the surface normal of a reflector, then the reflection BC is equivalent to:

$$\nabla \tau_{\text{out}}(\boldsymbol{X}(\boldsymbol{\sigma})) = \left(\boldsymbol{I} - 2\boldsymbol{\nu}_r \boldsymbol{\nu}_r^\top\right) \nabla \tau_{\text{in}}(\boldsymbol{X}(\boldsymbol{\sigma})). \tag{4.4}$$

That is, for a specular reflection to occur, the ingoing and outgoing ray directions should be equal, noting that continuity of τ requires $\|\nabla \tau_{\text{out}}(\boldsymbol{x}_0)\| = \|\nabla \tau_{\text{in}}(\boldsymbol{x}_0)\|$. Likewise, if $\boldsymbol{t}_e$ is the tangent vector of a diffracting edge, then the diffracted eikonal BCs simplify to:

$$\boldsymbol{t}_e^\top \nabla \tau_{\text{in}}(\boldsymbol{X}(\boldsymbol{\sigma})) = \boldsymbol{t}_e^\top \nabla \tau_{\text{out}}(\boldsymbol{X}(\boldsymbol{\sigma})). \tag{4.5}$$

Indeed, these conditions hold for a smoothly varying surface as well, but in that case $\boldsymbol{\nu}_r$ and $\boldsymbol{t}_e$ vary from point to point.

We come to an important point. Conditions (4.4) and (4.5) (which we think of as the specular reflection condition and Keller cone condition, respectively) tell us when a ray satisfies the generalized Fermat principle of Section 1.6. When we solve (4.3), we will compute τ throughout Ω, but, in general, (4.4) and (4.5) will not hold at each point $\boldsymbol{x} \in \Omega$. The points where these conditions fail to hold will have been reached by rays which have undergone diffraction. Likewise, when we solve the initial point source problem given by (4.2), if Ω is nonconvex, then some points will be reached by diffracted rays. These points lie in the shadow zone. Rays which violate these condition will be said to be unphysical.

One approach to dealing with physical rays would be to restrict the solution by trying to parametrize the shadow boundary, or the boundaries separating the points

in a reflected or diffracted field which satisfy (4.4) or (4.5) from those that don't (the physical rays). This quickly becomes complicated, and is hard to get right. We use a simpler approach based on tapering the amplitude of an unphysical ray using a scaling factor which depends on how much these conditions are violated. We discuss the details of how to do this after introducing some preliminary tools in Section 4.11.

4.3 Tetrahedron meshes: data structures and algorithms

As we mentioned, we will assume that $\partial\Omega \subseteq \mathbb{R}^3$, the boundary of our domain, is provided as a piecewise linear complex (PLC) [34]; that is, the boundary should consist of planar facets, its connected components should be manifold, and its facets should not have degenerate contact points between them. We will let the mesh size parameter be denoted by $h > 0$ and assume that the mesh is uniform: each of the cells will be of roughly the same size, so that as $h \to 0$, we can expect the number of nodes in the mesh to grow as $O(h^{-3})$. We denote the vertex set of the tetrahedron mesh by $\mathcal{V}_h \subseteq \Omega$, the edge set by $\mathcal{E}_h \subseteq \{1, \ldots, |\mathcal{V}_h|\}^2$, the set of faces by $\mathcal{F}_h \subseteq \{1, \ldots, |\mathcal{V}_h|\}^3$, and the set of cells/tetrahedra by $\mathcal{C}_h \subseteq \{1, \ldots, |\mathcal{V}_h|\}^4$. We also let $\partial\mathcal{V}_h$, $\partial\mathcal{E}_h$, and $\partial\mathcal{F}_h$ denote the boundary vertices, edges, and faces, respectively.

Useful information can be quickly and easily inferred from a tetrahedron mesh stored as an array of vertices and cells. For example, we can determine whether a face is on the boundary by checking if there is only one cell in $\mathcal{C}_h$ which contains it. Along the same lines, we can easily compute internal and external angles at a boundary edge by summing dihedral angles of incident tetrahedra. We will assume that the pair $(\mathcal{V}_h, \mathcal{C}_h)$ is such that it is possible to build a mesh data structure that allows us to look up neighborhood information in $O(1)$ time per query. In particular, if $\texttt{nb}(\boldsymbol{x})$ is the set of vertex neighbors of a point $\boldsymbol{x} \in \mathcal{V}_h$, then we assume that $|\texttt{nb}(\boldsymbol{x})| = O(1)$

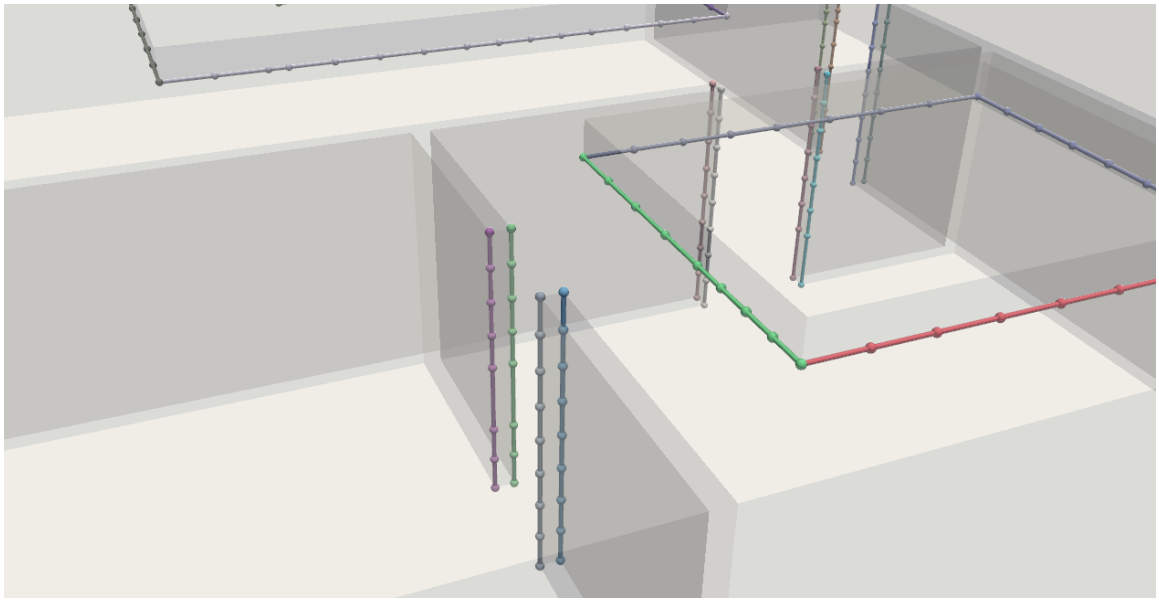


Figure 4.1: A close-up view of our test building problem (see Section 4.12) with the diffractors highlighted. Each diffractor is plotted using a different color. In the rendering, we visually distinguish between the different diffracting edges that comprise a diffractor.

independent of h. There are a variety of options for mesh data structures which support these operations, such as CGAL [47] and OpenVolumeMesh [76]. We elected to implement our own in our prototype code. Our motivation for doing so was to develop an understanding of how best to implement the mesh traversal algorithms required by this solver efficiently.

Diffractors and reflectors. We assume that $\partial\Omega$ is a PLC—but more than that, we assume that $\partial\Omega$ consists of a relatively small number of planar facets. This allows us to automatically determine the diffractors and reflectors in the scene. For each edge in $\partial\mathcal{E}_h$, if the interior angle of the edge is greater than π, we consider it to be a diffracting edge. We merge the diffracting edges into connected subsets of collinear edges using a breadth-first search. This results in the set of diffractors. Likewise, we collect the faces in $\partial\mathcal{F}_h$ into connected subsets of coplanar facets, resulting in the set of reflectors. See Figure 4.1 for the result of this process applied to our test mesh.

(Our test problems are discussed in more detail in Section 4.12.)

4.4 Semi-Lagrangian updates

The computational kernel of our multiple arrival solver is a JMM for solving the eikonal equation with $c \equiv 1$ on a tetrahedron mesh. The basic unit of work is the semi-Lagrangian update, as discussed elsewhere in this dissertation. In this section, we describe how line, triangle, and tetrahedron updates are done, and how to determine when we should commit the values of these updates. Recovering $\nabla T(\hat{\boldsymbol{x}})$ from these updates is straightforward. Computing $\nabla^2 T(\hat{\boldsymbol{x}})$ is simple for a line update. For triangle and tetrahedron updates, the situation is more complicated. We discuss these cases together in Section 4.5.

Line updates

In the general case, line updates are the solution of two-point BVPs. These are useful when updating from a point source, and can be used to initialize τ to high order in a ball surrounding $\boldsymbol{x}_{\text{src}}$. For example, we could discretize and numerically integrate the Euler-Lagrange equations for (1.4), given by (1.16), using standard methods for solving two-point BVPs [7]. If $\hat{\boldsymbol{x}}$ is the point being updated, and $\boldsymbol{x}$ is the base of the update, since we assume $c \equiv 1$ WLOG, we have:

$$T(\hat{\boldsymbol{x}}) = \|\hat{\boldsymbol{x}} - \boldsymbol{x}\|. \tag{4.6}$$

For $\nabla T(\hat{\boldsymbol{x}})$ and $\nabla^2 T(\hat{\boldsymbol{x}})$, we take the gradient of (4.6) with respect to $\hat{\boldsymbol{x}}$ twice:

$$\nabla T(\hat{\boldsymbol{x}}) = \frac{\hat{\boldsymbol{x}} - \boldsymbol{x}}{\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|}, \qquad \nabla^2 T(\hat{\boldsymbol{x}}) = \frac{1}{\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|}\left(\boldsymbol{I} - \frac{\hat{\boldsymbol{x}} - \boldsymbol{x}}{\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|} \otimes \frac{\hat{\boldsymbol{x}} - \boldsymbol{x}}{\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|}\right). \tag{4.7}$$

If c is constant but $c \neq 1$, we can recover the correct values of $T(\hat{\boldsymbol{x}}), \nabla T(\hat{\boldsymbol{x}})$, and $\nabla^2 T(\hat{\boldsymbol{x}})$ by scaling each of (4.6) and (4.7) by $1/c$.

Triangle updates

For a triangle update, we assume that the nodes $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ are `valid`, and that we have a polynomial $\mathsf{T}(\lambda)$ (where $\lambda \in [0,1]$) approximating T over $[\boldsymbol{x}_0, \boldsymbol{x}_1]$, which we parametrize as $\boldsymbol{x}_\lambda = \boldsymbol{x}_0 + \lambda(\boldsymbol{x}_1 - \boldsymbol{x}_0)$. As in Chapter 3, we denote local polynomial approximations using an uppercase serif font to distinguish them from global (piecewise) polynomials; e.g., T for the numerical eikonal.

To obtain T, there are two important cases to handle:

- When value of ∇T is finite on $[\boldsymbol{x}_0, \boldsymbol{x}_1]$ and we have both $T(\boldsymbol{x}_i)$ and $\nabla T(\boldsymbol{x}_i)$ available for $i = 0, 1$, we set T to be the cubic Hermite interpolant for the data $T(\boldsymbol{x}_0), T(\boldsymbol{x}_1), (\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \nabla T(\boldsymbol{x}_0)$, and $(\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \nabla T(\boldsymbol{x}_1)$.

- If $[\boldsymbol{x}_0, \boldsymbol{x}_1]$ is incident on a diffractor acting as a line source, the value of ∇T will be singular on $[\boldsymbol{x}_0, \boldsymbol{x}_1]$. Since T_{in} will be available over $[\boldsymbol{x}_0, \boldsymbol{x}_1]$, we set T to be the cubic polynomial interpolating the data $T_{\text{in}}(\boldsymbol{x}_0), T_{\text{in}}(\boldsymbol{x}_{1/3}), T_{\text{in}}(\boldsymbol{x}_{2/3})$, and $T_{\text{in}}(\boldsymbol{x}_1)$ using Lagrange interpolation.

Once we have computed T, we can compute $\nabla T(\hat{\boldsymbol{x}})$ from:

$$\nabla T(\hat{\boldsymbol{x}}) = \min_{0 \le \lambda \le 1} \Big\{ \mathsf{T}(\lambda) + \|\hat{\boldsymbol{x}} - \boldsymbol{x}_\lambda\| \Big\}. \tag{4.8}$$

This can be done very easily using a hybrid rootfinder [126, 23]. Denote the constrained optimum of (4.8) by λ^*. If λ^* is an interior point minimizer, we have:

$$(\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \nabla T(\boldsymbol{x}_{\lambda^*}) = (\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \frac{\hat{\boldsymbol{x}} - \boldsymbol{x}_{\lambda^*}}{\|\hat{\boldsymbol{x}} - \boldsymbol{x}_{\lambda^*}\|}. \tag{4.9}$$

Equation (4.9) gives us an optimality condition that relates the angles that the rays entering and exiting $\boldsymbol{x}_{\lambda^*}$ make with the line segment $[\boldsymbol{x}_0, \boldsymbol{x}_1]$. If $\boldsymbol{x}_{\lambda^*}$ is in free space (i.e., the interior of Ω), then this condition alone is insufficient to determine whether the ray is physical. We additionally need to ensure that both rays lie in the same plane. For this reason, we only do triangle updates when the updates are fully

immersed in $\partial\Omega$, and from diffracting edges (for all kinds of fields, not just edge-diffracted fields). If the update is fully immersed in a planar subset of $\partial\Omega$, we can confidently accept a ray computed from an interior point minimizer of (4.8). For an edge-diffracted triangle update, (4.9) is equivalent to (4.5), so we can be confident that these rays are physical, as well.

If $\lambda^* = 0$ or $\lambda^* = 1$, then (4.9) may not hold. However, it may not hold because the local ray has been unphysically diffracted, or because of a tolerable amount of numerical error. We take a simple-minded approach to handling this issue. First, we note that as we run our solver, we will encounter chains of adjacent triangle updates corresponding to the individual segments into which a diffractor has been discretized. Then, when $\boldsymbol{x}_{\lambda^*}$ is on the boundary of $[\boldsymbol{x}_0, \boldsymbol{x}_1]$, we first check whether $\boldsymbol{x}_{\lambda^*}$ corresponds to a terminal point on the diffractor: that is, whether or not there's another segment belonging to the diffractor which is also incident on $\boldsymbol{x}_{\lambda^*}$. If there is (i.e., $\boldsymbol{x}_{\lambda^*}$ is *not* a terminal point), then we do both updates and check whether they yield the same minimizer. If they do, we accept the update. If it *is* a terminal point, then we accept the update unconditionally. This causes a terminal point on the diffractor to behave as a point source, which is correct. To avoid doing an unnecessary number of triangle updates (and tetrahedron updates—see the next section), we cache "old updates" for later use. This is explained in Section 4.6.

Tetrahedron updates

Tetrahedron updates require us to compute a polynomial approximation of T over $\mathrm{conv}(\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2)$, where $\boldsymbol{x}_0, \boldsymbol{x}_1$, and $\boldsymbol{x}_2$ are the `valid` update points. We T to be the triangular cubic Hermite polynomial interpolating the data:

$$T(\boldsymbol{x}_i), \quad (\boldsymbol{x}_j - \boldsymbol{x}_i)^\top \nabla T(\boldsymbol{x}_i), \quad (\boldsymbol{x}_k - \boldsymbol{x}_i)^\top \nabla T(\boldsymbol{x}_i), \quad i, j, k \in \{0, 1, 2\}, \tag{4.10}$$

and such that i, j, and k are distinct. Note that a bivariate polynomial has 10 degrees of freedom (DOFs), while we only have 9 pieces of data available. Bernstein-Bézier interpolation provides useful geometric intuition for the interpolation problem. The "nine-parameter interpolant" is a standard cubic Hermite interpolant defined on a triangle which is $O(h^3)$ accurate and requires the 9 DOFs that we have available [36]. The 10th DOF is eliminated using condensation of parameters [49, 78].

In order to define the minimization problem for a tetrahedron update, we define the convex coefficients λ_0, λ_1, and λ_2 so that $\lambda_0 = 1 - \lambda_1 - \lambda_2$ and $\lambda_i \geq 0$, and $\sum_i \lambda_i = 1$. We let $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$. Here, we define T over $\boldsymbol{\lambda}$, but we note that Bézier triangles are typically defined over $(\lambda_0, \lambda_1, \lambda_2)$. Finally, we let $\boldsymbol{x}_{\boldsymbol{\lambda}} = \boldsymbol{x}_0 + \lambda_1(\boldsymbol{x}_1 - \boldsymbol{x}_0) + \lambda_2(\boldsymbol{x}_2 - \boldsymbol{x}_0)$. We also let Δ^2 denote the set of vectors $\boldsymbol{\lambda}$ (as in Chapter 2). Then, we have:

$$T(\hat{\boldsymbol{x}}) = \min_{\boldsymbol{\lambda} \in \Delta^2} \left\{ \mathsf{T}(\boldsymbol{\lambda}) + \|\hat{\boldsymbol{x}} - \boldsymbol{x}_{\boldsymbol{\lambda}}\| \right\}. \tag{4.11}$$

This is a two-dimension constrained minimization problem with three inequality constraints: $\lambda_i \geq 0$ for $i = 0, 1, 2$. The domain is the standard unit triangle in $\mathbb{R}^2$.

To solve (4.11), we use an active set method, which is a type of projected Newton's method [94, 17]. The basic idea is to generate a sequence of iterates $\boldsymbol{\lambda}_k$ $(k = 0, 1, \ldots)$ by repeatedly minimizing a local quadratic model of the cost function. Let $f(\boldsymbol{\lambda}) = \mathsf{T}(\boldsymbol{\lambda}) + \|\hat{\boldsymbol{x}} - \boldsymbol{x}_{\boldsymbol{\lambda}}\|$. Let $\delta\boldsymbol{\lambda} = \boldsymbol{\lambda}_{k+1} - \boldsymbol{\lambda}_k$, and Taylor expand:

$$f(\boldsymbol{\lambda}_{k+1}) = f(\boldsymbol{\lambda}_k + \delta\boldsymbol{\lambda}) = f(\boldsymbol{\lambda}_k) + \nabla f(\boldsymbol{\lambda}_k)^\top \delta\boldsymbol{\lambda} + \frac{1}{2}\delta\boldsymbol{\lambda}^\top \nabla^2 f(\boldsymbol{\lambda}_k)\delta\boldsymbol{\lambda} + O(\|\delta\boldsymbol{\lambda}\|^3). \tag{4.12}$$

To compute $\boldsymbol{\lambda}_{k+1}$, we solve:

$$\delta\boldsymbol{\lambda}^* = \arg \min_{\boldsymbol{\lambda}_k + \delta\boldsymbol{\lambda} \in \Delta^2} \left\{ f(\boldsymbol{\lambda}_k) + \nabla f(\boldsymbol{\lambda}_k)^\top \delta\boldsymbol{\lambda} + \frac{1}{2}\delta\boldsymbol{\lambda}^\top \nabla^2 f(\boldsymbol{\lambda}_k)\delta\boldsymbol{\lambda} \right\}. \tag{4.13}$$

This gives us a tentative guess for $\boldsymbol{\lambda}_{k+1}$. To ensure convergence, we can then use an inexact backtracking line search to approximately minimize f over $[\boldsymbol{\lambda}_k, \boldsymbol{\lambda}_{k+1}]$. We do not dwell on the details of this optimization algorithm, but because of the low dimension, it is possible to program it efficiently. We use the same approach

in Chapter 2. The smoothness of f ensures that the number of iterations of the active set method is small, typically requiring fewer than 5 iterations for convergence to machine precision. The inequality-constrained quadratic program whose solution gives $\boldsymbol{\lambda}_{k+1}$ can be solved exactly without using an iterative method.

After we solve (4.11), we again need to consider the cases where $\boldsymbol{\lambda}^*$ is an interior point minimizer and a boundary minimizer separately. In the former case, letting:

$$\delta \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{x}_0 & \boldsymbol{x}_2 - \boldsymbol{x}_0 \end{bmatrix} \in \mathbb{R}^{3\times 2}, \tag{4.14}$$

we have:

$$\delta \boldsymbol{X}^\top \nabla T(\boldsymbol{x}_{\boldsymbol{\lambda}^*}) = \delta \boldsymbol{X}^\top \frac{\hat{\boldsymbol{x}} - \boldsymbol{x}_{\boldsymbol{\lambda}^*}}{\|\hat{\boldsymbol{x}} - \boldsymbol{x}_{\boldsymbol{\lambda}^*}\|}. \tag{4.15}$$

We can readily see that this optimality condition tells us that when $\boldsymbol{\lambda}^*$ lies in the interior of Δ^2, the angle that the ray entering and exiting $\boldsymbol{x}_{\lambda^*}$ makes with the plane spanned by the base of the update must be the same. This makes sense: this is the same as requiring that the ray does not bend unphysically as it passes through the base of the update. Since c is smooth over the update base, there should be no reason for the ray to spuriously bend in free space.

If $\boldsymbol{\lambda}^*$ is a boundary point, then we find ourselves in the same situation as before, when we needed to decide whether to accept a triangle update with a boundary minimizer. If the Lagrange multipliers of the problem are nonzero, then the ray is bending. If this happens along an edge which is embedded in the boundary, this is fine; but if it happens in free space, we need to determine whether the amount of bending is a tolerable numerical artifact or not. To do this, we follow the same procedure as before. We keep track of adjacent tetrahedron updates using the approach outlined in Section 4.6. For a boundary point that lies in the interior of one of three edges of Δ^2, if we find two adjacent tetrahedron updates that yield the same optimum $\boldsymbol{x}_{\lambda^*}$, we accept the update. If $\boldsymbol{\lambda}^*$ is a vertex of Δ^2, we look for three. See Figure 4.2.

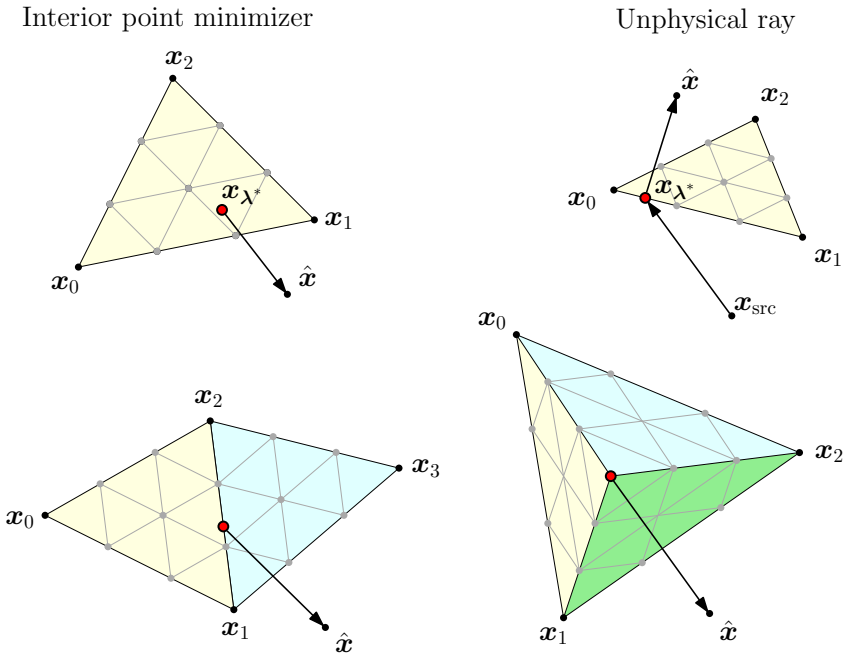


Figure 4.2: A depiction of the four cases that we can come across while doing updates. *Top left*: An interior point minimizer. If the predicted $T(\hat{\boldsymbol{x}})$ is smaller than the current value for $T(\hat{\boldsymbol{x}})$, this ray can be committed. *Top right:* An "unphysical ray", where diffraction has occurred spuriously in free space, unless $[\boldsymbol{x}_0, \boldsymbol{x}_1] \subseteq \partial\Omega$. We do not accept these updates. *Bottom left:* A physical ray due to a boundary minimizer shared by two adjacent updates, and (*bottom right*) three adjacent updates. These updates might be encountered during different calls to `update_neighbors`, forcing us to keep track of old updates. See Sections 4.6 and 4.7.

Nonzero Lagrange multipliers for shared optima

We briefly discuss why we can expect to encounter boundary minimizers shared by adjacent updates. We will consider the situation in 2D, since no essential details are lost. Consider a pair of triangle updates $(\hat{\boldsymbol{x}}, \boldsymbol{x}_0, \boldsymbol{x}_1)$ and $(\hat{\boldsymbol{x}}, \boldsymbol{x}_0, \boldsymbol{x}_1')$ with a point source $\boldsymbol{x}_{\text{src}}$ as shown in Figure 4.3, with $c \equiv 1$. We will restrict our attention to the update $(\hat{\boldsymbol{x}}, \boldsymbol{x}_0, \boldsymbol{x}_1)$ without loss of generality. The cost function for this update can be written:

$$f(\lambda) = T(\lambda) + \|\hat{\boldsymbol{x}} - \boldsymbol{x}_\lambda\| \tag{4.16}$$

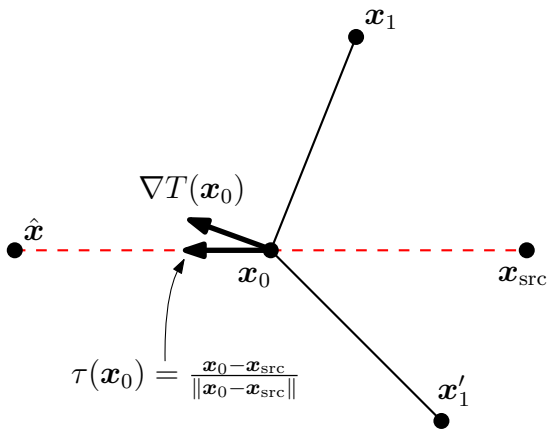


Figure 4.3: A pair of triangle updates which should yield a shared optima, and where each update's Lagrange multipliers are both zero if $T(\boldsymbol{x}_0)$ and $\nabla T(\boldsymbol{x}_0)$ are known exactly. If there is error in the data at $\boldsymbol{x}_0$, then these updates can continue to produce a shared minimizer on the boundary, but for which the corresponding Lagrange multiplier is no longer equal to zero. This motivates the use of the update lists and cached updates described in Section 4.6.

where $\boldsymbol{x}_\lambda = \boldsymbol{x}_0 + \lambda(\boldsymbol{x}_1 - \boldsymbol{x}_0)$, and where we assume that T is the following cubic Hermite polynomial:

$$\begin{aligned} T(\lambda) = \tau(\boldsymbol{x}_0)H_0(\lambda) + \tau(\boldsymbol{x}_1)H_1(\lambda) \\ + (\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \nabla\tau(\boldsymbol{x}_0)K_0(\lambda) + (\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \nabla\tau(\boldsymbol{x}_1)K_1(\lambda). \end{aligned} \tag{4.17}$$

where $\tau(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{x}_{\text{src}}\|$ and $\nabla\tau(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{x}_{\text{src}})/\|\boldsymbol{x} - \boldsymbol{x}_{\text{src}}\|$. The functions H_0, H_1, K_0, and K_1 are the cardinal basis functions for cubic Hermite interpolation over $[0, 1]$, satisfying:

$$\begin{bmatrix} H_0(0) & H_1(0) & K_0(0) & K_1(0) \\ H_0(1) & H_1(1) & K_0(1) & K_1(1) \\ H_0'(0) & H_1'(0) & K_0'(0) & K_1'(0) \\ H_0'(1) & H_1'(1) & K_0'(1) & K_1'(1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.18}$$

The minimization problem to solve for this update is:

$$\begin{aligned} \text{minimize} \quad & f(\lambda) \\ \text{subject to} \quad & 0 \le \lambda \le 1 \end{aligned} \tag{4.19}$$

The Lagrangian is:

$$L(\lambda, \alpha_0, \alpha_1) = f(\lambda) + \lambda\alpha_0 + (1-\lambda)\alpha_1. \tag{4.20}$$

It's clear that with the setup as shown, the minimizer for either of the triangle updates is $\lambda = 0$, which is a constrained optimum. The KKT conditions then tell us that a constrained optimum satisfies:

$$\frac{\partial L}{\partial \lambda} = f'(\lambda) + \alpha_0 - \alpha_1 = 0. \tag{4.21}$$

For $\lambda = 0$, the Lagrange multiplier corresponding to the inequality constraint $\lambda \leq 1$ will be zero; that is, $\alpha_1 = 0$ if $\lambda \neq 1$. Hence, for this minimizer:

$$\alpha_0 = -f'(0) = (\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \frac{\hat{\boldsymbol{x}} - \boldsymbol{x}_0}{\|\hat{\boldsymbol{x}} - \boldsymbol{x}_0\|} - T'(0). \tag{4.22}$$

Now, we have from (4.18):

$$T'(0) = (\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \nabla\tau(\boldsymbol{x}_0) = (\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \frac{\boldsymbol{x}_0 - \boldsymbol{x}_{\text{src}}}{\|\boldsymbol{x}_0 - \boldsymbol{x}_{\text{src}}\|}. \tag{4.23}$$

Hence:

$$\alpha_0 = (\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \frac{\hat{\boldsymbol{x}} - \boldsymbol{x}_0}{\|\hat{\boldsymbol{x}} - \boldsymbol{x}_0\|} - (\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \frac{\boldsymbol{x}_0 - \boldsymbol{x}_{\text{src}}}{\|\boldsymbol{x}_0 - \boldsymbol{x}_{\text{src}}\|}. \tag{4.24}$$

We can conclude from this that the Lagrange multiplier α_0 will be zero only if:

$$\frac{\boldsymbol{x}_1 - \boldsymbol{x}_0}{\|\boldsymbol{x}_1 - \boldsymbol{x}_0\|}^\top \frac{\hat{\boldsymbol{x}} - \boldsymbol{x}_0}{\|\hat{\boldsymbol{x}} - \boldsymbol{x}_0\|} = \frac{\boldsymbol{x}_1 - \boldsymbol{x}_0}{\|\boldsymbol{x}_1 - \boldsymbol{x}_0\|}^\top \frac{\boldsymbol{x}_0 - \boldsymbol{x}_{\text{src}}}{\|\boldsymbol{x}_0 - \boldsymbol{x}_{\text{src}}\|}. \tag{4.25}$$

This is an angle condition: the angle that the ray leading from $\boldsymbol{x}_{\text{src}}$ to $\boldsymbol{x}_0$ makes with the update integral should be the same as the ray leading from the update point $\hat{\boldsymbol{x}}$ to $\boldsymbol{x}_0$. For the picture shown, this is evidently always the case.

If we replace the exact dat with inexact data so that:

$$T = T(\boldsymbol{x}_0)H_0 + T(\boldsymbol{x}_1)H_1 + (\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \nabla T(\boldsymbol{x}_0)K_0 + (\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \nabla T(\boldsymbol{x}_1)K_1 \tag{4.26}$$

then we instead get:

$$\alpha_0 = (\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \frac{\hat{\boldsymbol{x}} - \boldsymbol{x}_0}{\|\hat{\boldsymbol{x}} - \boldsymbol{x}_0\|} - (\boldsymbol{x}_1 - \boldsymbol{x}_0)\nabla T(\boldsymbol{x}_0) \tag{4.27}$$

With the same equation with $\boldsymbol{x}_1'$ substituted for $\boldsymbol{x}_1$ for the opposite triangle update. For our simple model, we expect $\nabla T(\boldsymbol{x}_0)$ to satisfy:

$$\nabla T(\boldsymbol{x}_0) = \frac{\boldsymbol{x}_0 - \boldsymbol{x}_{\text{src}}}{\|\boldsymbol{x}_0 - \boldsymbol{x}_{\text{src}}\|} + O(h^p), \tag{4.28}$$

for some p. For a small enough error, we can expect each update to continue to produce $\lambda = 0$ as a constrained minimizer. However, in this case, the angle condition will be violated for both updates and α_1 will no longer be zero. So, in these cases, we can no longer use the Lagrange multiplier test to check for an interior point update, and resort to our algorithm. The Lagrange multiplier will be small, and we could consider accepting an update depending in the magnitude of the Lagrange multiplier so that the resulting errors are compatible with the expected global error of the solver, but the proposed approach avoids the complication of introducing any additional parameters which might need to be tuned.

4.5 Propagating the Hessian of the eikonal

From our simplified paraxial ray theory described in Section 1.5, if we write $\nabla^2 T(\boldsymbol{\varphi}(\sigma)) = \boldsymbol{P}(\sigma)\boldsymbol{Q}(\sigma)^{-1}$, we have the following system of ODEs:

$$\frac{d\boldsymbol{P}}{d\sigma} = \frac{1}{2s(\boldsymbol{x})}\nabla^2\Big(s(\boldsymbol{x})^2\Big)\Big|_{\boldsymbol{x}=\boldsymbol{\varphi}(\sigma)} \boldsymbol{Q}(\sigma), \qquad \frac{d\boldsymbol{Q}}{d\sigma} = \frac{1}{s(\boldsymbol{\varphi}(\sigma))}\boldsymbol{P}(\sigma). \tag{4.29}$$

Since $s \equiv 1$, this simplifies to:

$$\frac{d\boldsymbol{P}}{d\sigma} \equiv 0, \qquad \frac{d\boldsymbol{Q}}{d\sigma} = \boldsymbol{P}(\sigma). \tag{4.30}$$

To integrate eq. (4.30) over a local ray, we let $\boldsymbol{Q}(0) = \boldsymbol{I}$ so that:

$$\boldsymbol{P}(\sigma) \equiv \boldsymbol{P}(0) = \nabla^2 T(\boldsymbol{\varphi}(\mathbf{0})) = \nabla^2 T(\boldsymbol{x}_{\boldsymbol{\lambda}}), \tag{4.31}$$

and then integrate:

$$\boldsymbol{Q}(\sigma) = \boldsymbol{Q}(0) + \int_0^L \boldsymbol{P}(\sigma)d\sigma = \boldsymbol{I} + L\boldsymbol{P}(0), \tag{4.32}$$

where $L = \|\hat{\boldsymbol{x}} - \boldsymbol{x}_{\boldsymbol{\lambda}}\|$. Then:

$$\nabla^2 T(\hat{\boldsymbol{x}}) = \boldsymbol{P}(0)\Big(\boldsymbol{I} + L\boldsymbol{P}(0)\Big) = \nabla^2 T(\boldsymbol{x}_\lambda)\Big(\boldsymbol{I} + L\nabla^2 T(\boldsymbol{x}_\lambda)\Big)^{-1}. \tag{4.33}$$

Equation (4.33) gives us a very simple means of propagating the Hessian along the update ray when we know the value of $\nabla^2 T(\boldsymbol{x}_\lambda)$, but this is not always the case. On caustics, the derivatives of τ are singular. At a point source, since $\tau(\boldsymbol{x}) \sim \|\boldsymbol{x} - \boldsymbol{x}_{\text{src}}\|$, we have a good local asymptotic model for the eikonal. This gives us our simple formula for the Hessian for a line update, discussed in Section 4.4 (line updates are only ever done from point sources).

The other case that arises is propagating the Hessian from a diffracting edge. In the case of a triangle update, we can do several triangle updates in order to compute a finite difference approximation to $\nabla^2 T(\hat{\boldsymbol{x}})$. If $\boldsymbol{e}_i \in \mathbb{R}^3$ is the ith standard basis vector, we set:

$$\nabla^2 T(\hat{\boldsymbol{x}})\boldsymbol{e}_i = \frac{\nabla T(\hat{\boldsymbol{x}} + \delta \boldsymbol{e}_i) - \nabla T(\hat{\boldsymbol{x}} - \delta \boldsymbol{e}_i)}{2\delta} + O(\delta^2), \qquad \delta > 0. \tag{4.34}$$

Setting $\delta = h^2$ is a reasonable choice, since this ensures that the perturbations about $\hat{\boldsymbol{x}}$ are small relative to the size of the update.

For a tetrahedron update, we will only encounter updates with one or two nodes $\boldsymbol{x}_i$ with a singular Hessian. If any of the Hessians are singular, the base of the update is incident on one or two diffractors. (We skip tetrahedron updates that are incident on a point source, since we can just do a line update from the point source.) For a given diffracted field, there will only be on diffractor over which $\nabla^2 \tau$ is singular. Then we can use the adjacent triangle updates to compute a finite difference approximation to $\nabla^2 T(\hat{\boldsymbol{x}})$.

4.6 Update lists and cached updates

We use a Dijkstra-like algorithm to march the values of $T, \nabla T$, and $\nabla^2 T$ throughout the domain. We will describe this algorithm in more detail shortly. When using a

Dijkstra-like (direct) solver for the eikonal equation, it is necessary to ensure causality. That is, the nodes $\hat{\boldsymbol{x}}$ should be marked `valid` in nondecreasing order of $T(\hat{\boldsymbol{x}})$. A sufficient condition for causality is that each update that is done nonobtuse. In general, for a particular update, the cone spanned by nonnegative combinations of $\boldsymbol{x}_i - \hat{\boldsymbol{x}}$ for each i such that $0 \leq i < n$, should be able to rotated so that it fits inside the nonnegative orthant. Equivalently, we should have:

$$\frac{(\boldsymbol{x}_i - \hat{\boldsymbol{x}})}{\|\boldsymbol{x}_i - \hat{\boldsymbol{x}}\|}^{\top} \frac{(\boldsymbol{x}_j - \hat{\boldsymbol{x}})}{\|\boldsymbol{x}_j - \hat{\boldsymbol{x}}\|} \geq 0, \qquad i \neq j. \tag{4.35}$$

Different methods of ensuring causality on an unstructured triangle mesh have been developed in the past, but these approaches are unwieldy in 3D, and while the geometric constructions presented in these works appears to be generalizable higher-dimensions, we are unsure whether this has been attempted [72, 119]. Instead, we use the optimality conditions discussed in Section 4.4 augmented with the data structures and algorithms discussed in this section. This avoids the need for any complicated geometric constructions, unlike previous approaches.

First, we note that when we accept a node $\boldsymbol{x}_0$ (we pop it from the priority queue and set its state to `valid`, thereby fixing the solution at this point), we will update nodes $\hat{\boldsymbol{x}}$ that are `trial` and which neighbor $\boldsymbol{x}_0$. Each time we update a node $\hat{\boldsymbol{x}}$, we will do a number of triangle and tetrahedron updates, which are described in Section 4.4. These individual updates are the smallest unit of work in our algorithm, but to actually compute a new value at $\hat{\boldsymbol{x}}$, we need to do several of them. We refer to the set of updates for the pair $(\hat{\boldsymbol{x}}, \boldsymbol{x}_0)$ as an update list. Specifically, upon accepting $\hat{\boldsymbol{x}}$ and visiting $\boldsymbol{x}_0$, we will be able to compute a set:

$$\texttt{update\_list}(\hat{\boldsymbol{x}}; \boldsymbol{x}_0) = \left\{ (\boldsymbol{x}_0^{(j)}, \ldots, \boldsymbol{x}_{n-1}^{(j)}) : j = 0, \ldots, n_{\text{up}} \right\}, \tag{4.36}$$

which enumerates each of the individual semi-Lagrangian updates needed to compute a new jet at $\hat{\boldsymbol{x}}$. Here, $n_{\text{up}} = n_{\text{up}}(\hat{\boldsymbol{x}}; \boldsymbol{x}_0)$ is the number of updates, j is the index of the update, and n is the dimension of the updates (i.e., $n = 1$ for a line update, $n = 2$ for

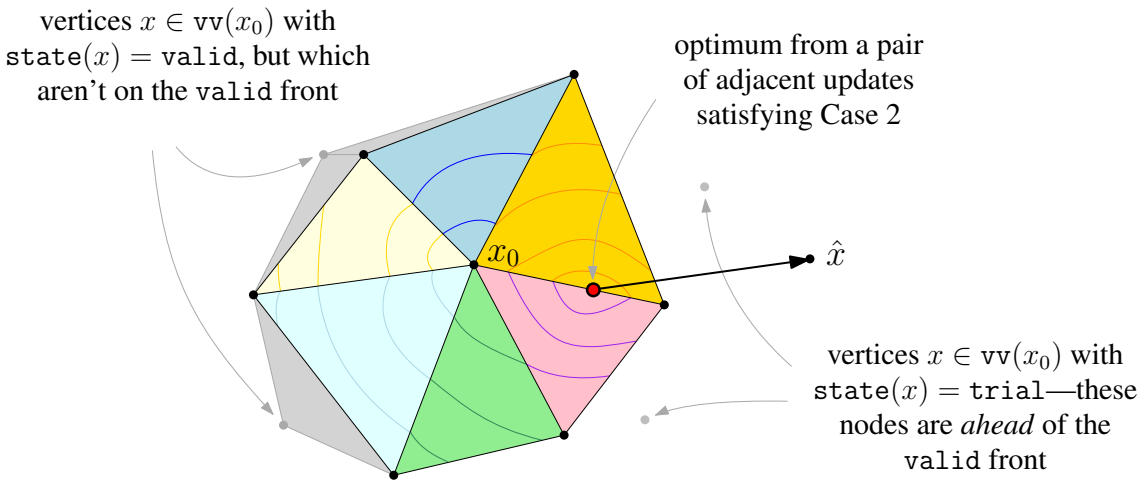


Figure 4.4: A depiction of the optimization over the fan of triangles on the `valid` front surrounding $\boldsymbol{x}_0$. For some context, we depict nearby nodes that are `valid` (behind the `valid` front), and `trial` (ahead of it). We sort the corresponding tetrahedron updates into $\texttt{update\_list}(\hat{\boldsymbol{x}}, \boldsymbol{x}_0)$ and check for a set of tetrahedron updates that satisfy the conditions depicted in Figure 4.2. In this example, we find a pair of adjacent tetrahedron updates with a shared boundary minimizer which can be accepted. We sketch some plausible level sets of the cost function for the minimization problem in (4.11). Over the entire triangle fan, these cost functions are only C^0 between triangles.

a triangle update, and $n = 3$ for a tetrahedron update). We treat updates of different dimension separately.

Now, to update $\hat{\boldsymbol{x}}$, we solve each update in $\texttt{update\_list}(\hat{\boldsymbol{x}}; \boldsymbol{x}_0)$, and then sort $\texttt{update\_list}(\hat{\boldsymbol{x}}; \boldsymbol{x}_0)$ in increasing order of $T(\hat{\boldsymbol{x}})$. Clearly, we want to accept the update with the smallest T value. However, as we discussed in Section 4.4, we may encounter updates with optima on the boundary. Sorting $\texttt{update\_list}(\hat{\boldsymbol{x}}; \boldsymbol{x}_0)$ lets us quickly check whether multiple updates with boundary minima yield the same ray—if this is the case, the value of $T(\hat{\boldsymbol{x}})$ for each update will be the same, and we can commit the corresponding jet. On the other hand, after processing $\texttt{update\_list}(\hat{\boldsymbol{x}}; \boldsymbol{x}_0)$ and committing a new value at $\hat{\boldsymbol{x}}$, there may be orphaned updates in the list which we need to keep track of. To this end, we define a set $\texttt{cached\_updates}(\hat{\boldsymbol{x}})$ for each $\hat{\boldsymbol{x}}$. This is a set of updates, like `update_list`, which is instantiated when $\hat{\boldsymbol{x}}$ is first inserted into the priority queue. After solving the updates in $\texttt{update\_list}(\hat{\boldsymbol{x}}; \boldsymbol{x}_0)$ for

some $\boldsymbol{x}_0$, we insert the unused updates with a boundary minimizer (the orphaned updates) into `cached_updates`$(\hat{\boldsymbol{x}})$. Each time we evaluate `update_list`$(\hat{\boldsymbol{x}};\ \cdot\ )$, we populate it with the current values of `cached_updates`$(\hat{\boldsymbol{x}})$. This allows us to keep track of updates at different phases in the algorithm, and also allows us to avoid re-solving old updates. Finally, once $\hat{\boldsymbol{x}}$ itself is popped from the priority queue and made `valid`, we delete `cached_updates`$(\hat{\boldsymbol{x}})$ to keep memory use modest.

4.7 Jet marching on a tetrahedron mesh

The outer loop of our tetrahedron mesh JMM is given by the following algorithm:

procedure `solve`
- Initialize BCs for each $\boldsymbol{x} \in \Gamma_h$
- Sort nodes with BCs into `front`
- Set `num_accepted` $\leftarrow 0$
- **while** `front` is not empty **do**
  - Set $\boldsymbol{x}_0 \leftarrow$ `pop(front)`
  - Set `state`$(\boldsymbol{x}_0) \leftarrow$ `valid`
  - **if** we aren't solving a point source problem **then**
    - Run `compute_`$\boldsymbol{t}_{\text{in}}(\boldsymbol{x}_0)$
  - Run `compute_`$\boldsymbol{t}_{\text{out}}(\boldsymbol{x}_0)$
  - Delete data structures used to track old updates for $\boldsymbol{x}_0$
  - **for** $\hat{\boldsymbol{x}} \in$ `nb`$(\boldsymbol{x}_0)$ such that `state`$(\hat{\boldsymbol{x}}) =$ `far` **do**
    - Merge $\hat{\boldsymbol{x}}$ into `front`
  - Run `update_neighbors`$(\boldsymbol{x}_0)$
  - Set `accepted`$[\boldsymbol{x}_0] \leftarrow$ `num_accepted`
  - Set `num_accepted` $\leftarrow$ `num_accepted` $+ 1$

This is a modified version of the usual Dijkstra-like marching algorithm. The key additions are data structures used to track old updates and tracking the order in which nodes are fixed in `accepted`. Additionally, the procedure `update_neighbors` is modified to generate the update lists discussed in Section 4.6 in a manner which

is compatible with solving the eikonal equation on a tetrahedron mesh. We also track each node's parent—the point in the base of the update from which a node was received its final value. Using this information, we approximately propagate a variety of auxiliary quantities along the characteristics of the solution to compute boundary conditions for the reflected and diffracted amplitude. This is described in Section 4.8.

The procedure `update_neighbors` has two parts. First, we find all diffracting edges incident on $\boldsymbol{x}_0$, and do triangle updates from these diffracting edges (which are now `valid`, and can therefore be updated from) to nearby `trial` nodes:

procedure `do_nearby_diff_edge_updates`($\boldsymbol{x}_0$)
  for $\boldsymbol{x}_1 \in \mathcal{V}_h$ s.t. $(\boldsymbol{x}_0, \boldsymbol{x}_1)$ is a diffracting edge and `state`($\boldsymbol{x}_1$) = `valid` **do**
    for $\hat{\boldsymbol{x}} \in$ `nb`($\boldsymbol{x}_0$) $\cup$ `nb`($\boldsymbol{x}_1$) s.t. `state`($\hat{\boldsymbol{x}}$) = `trial` **do**
      Do the triangle update $(\hat{\boldsymbol{x}}, \boldsymbol{x}_0, \boldsymbol{x}_1)$
      if the update ray is physical and $T(\hat{\boldsymbol{x}})$ is smaller **then**
        Update the values for $T(\hat{\boldsymbol{x}}), \nabla T(\hat{\boldsymbol{x}}), \nabla^2 T(\hat{\boldsymbol{x}})$
        Adjust $\hat{\boldsymbol{x}}$'s position in `front`

Afterwards, we update each `trial` node neighboring $\boldsymbol{x}_0$. In order to do so, we find the fan of triangles on the `valid` front which are incident on $\boldsymbol{x}_0$. See Figure 4.4. We then form an update list from these triangles, sort them, and commit them as described in Section 4.6:

procedure `update`($\hat{\boldsymbol{x}}$, $\boldsymbol{x}_0$)
  Set `update_list`($\hat{\boldsymbol{x}}, \boldsymbol{x}_0$) $\leftarrow \{\}$
  for $C \in \mathcal{C}_h$ s.t. $\boldsymbol{x}_0 \in C$ **do**
    if C consists of exactly three `valid` nodes **then**
      Let $\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2$ be the `valid` nodes in C
      Add $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2\}$ to `update_list`($\hat{\boldsymbol{x}}, \boldsymbol{x}_0$)
  Solve each update in `update_list`($\hat{\boldsymbol{x}}, \boldsymbol{x}_0$)
  Add all updates from `cached_updates`($\hat{\boldsymbol{x}}$) to `update_list`($\hat{\boldsymbol{x}}, \boldsymbol{x}_0$)
  Set `cached_updates`($\hat{\boldsymbol{x}}$) $\leftarrow \{\}$
  Sort `update_list`($\hat{\boldsymbol{x}}, \boldsymbol{x}_0$) in increasing order of $T(\hat{\boldsymbol{x}})$

if `update_list` has a run of updates producing a physical ray **then**
  if $T(\hat{\boldsymbol{x}})$ is smaller than the current value of $T(\hat{\boldsymbol{x}})$ **then**
    Update the values for $T(\hat{\boldsymbol{x}}), \nabla T(\hat{\boldsymbol{x}}), \nabla^2 T(\hat{\boldsymbol{x}})$
    Adjust $\hat{\boldsymbol{x}}$'s position in `front`
Add any orphaned updates to `cached_updates`$(\hat{\boldsymbol{x}})$

The way in which determine whether a set of updates produces a physical ray (and how we extract this information from `update_list`$(\hat{\boldsymbol{x}}, \boldsymbol{x}_0)$ once it's been sorted), and how to determine whether an update is orphaned, is explained in Section 4.6.

While doing tetrahedron updates $(\hat{\boldsymbol{x}}, \boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2)$ from the update fan, we will occasionally encounter diffracting edges corresponding to $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ that will not have been discovered by `do_nearby_diff_edge_updates`. We have found that "snapping" the ray to this diffracting edge by doing the triangle update $(\hat{\boldsymbol{x}}, \boldsymbol{x}_1, \boldsymbol{x}_2)$ increases the quality of the solution in the shadow zone just past a diffracting edge. (We note that restricting the domain of the tetrahedron update to $[\boldsymbol{x}_1, \boldsymbol{x}_2]$ results in the same optimization problem as the one solved by the triangle update $(\hat{\boldsymbol{x}}, \boldsymbol{x}_1, \boldsymbol{x}_2)$. This generally results in a minimizer with a larger value. However, it avoids inexact rays due to numerical error.)

4.8 Transporting auxiliary quantities

When we solve the eikonal equation, each time we update a node $\hat{\boldsymbol{x}}$ using a semi-Lagrangian update, we determine the nodes comprising the base of the update and parametrize the point in the update base from which the local ray reaches $\hat{\boldsymbol{x}}$. See Figure 4.5. We also parametrize the ray itself, leading from $\boldsymbol{x}_{\boldsymbol{\lambda}}$ to $\hat{\boldsymbol{x}}$. In Chapter 3, this also meant obtaining a local polynomial approximation of the ray, which we denoted $\boldsymbol{\varphi}$. We assume that a node has n_{par} parents ($0 \leq n_{\text{par}} \leq 3$ for a tetrahedron mesh), and let $\boldsymbol{\lambda}$ denote the vector of convex coefficients yielding $\boldsymbol{x}_{\boldsymbol{\lambda}}$. Then, we define:

$$\texttt{parent}(\hat{\boldsymbol{x}}) = \Big((\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n_{\text{par}}-1}), \boldsymbol{\lambda} \Big). \tag{4.37}$$

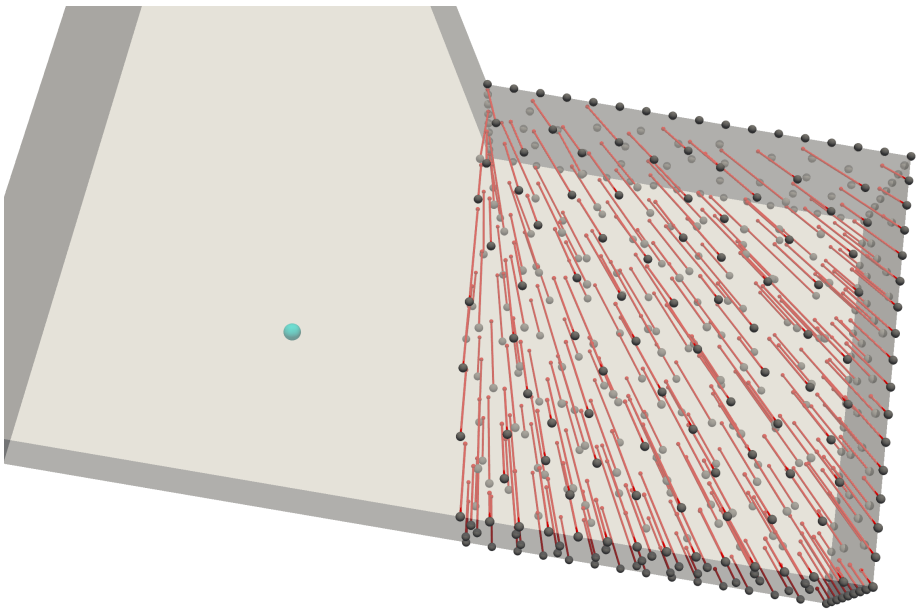


Figure 4.5: A reflection from a surface in a wedge-shaped room. Solving the eikonal equation results in a precomputed dynamic programming plan if we keep track of the order in which nodes are accepted and the parent of each node. Here, we show the original point source as a cyan sphere, and connect each node $\hat{\boldsymbol{x}}$ to its parent $\texttt{parent}(\hat{\boldsymbol{x}})$ with a red edge. We can see how the edges follow the characteristics of the eikonal.

If $\hat{\boldsymbol{x}}$ has BCs, then we let $\texttt{parent}(\hat{\boldsymbol{x}})$ be undefined—that is, $\hat{\boldsymbol{x}}$ has no parent.

At the same time, as we run our solver, we maintain an array `accepted` of length $|\mathcal{V}_h|$ which indicates the order in which each node $\hat{\boldsymbol{x}}$ was removed from the priority queue, with its eikonal value subsequently fixed. Since we index each vertex $\boldsymbol{x} \in \mathcal{V}_h$ using an integer index, when the mth node popped from the priority queue is the lth vertex, we set $\texttt{accepted}[m] \leftarrow l$.

We can see that with `parent` and `accepted` so defined, if $\hat{\boldsymbol{x}}$ is the lth vertex, and if $l_0, \dots, l_{n_{\text{par}}-1}$ are the indices of the parent nodes of $\hat{\boldsymbol{x}}$, and $m_0, \dots, m_{n_{\text{par}}-1}$ are such that $\texttt{accepted}[m_i] = l_i$ $(i = 0, \dots, n_{\text{par}} - 1)$, then we can immediately conclude that $m_i < m$ for $i = 0, \dots, n_{\text{par}} - 1$. There is nothing complicated about this statement: all it says is that a node's parents will have been accepted before the node itself.

We think of `parent` and `accepted` together constituting the dynamic programming plan used to compute T. By storing `parent` and `accepted`, we are able to cache

this dynamic programming plan so that we can reuse it later. This is useful, because it enables an $O(N)$ algorithm for approximately transporting quantities defined on Γ_h along the characteristics of (1.4). This is crucial for being able to apply the BCs for the reflected and diffracted amplitude, along with the scaling factors used to taper the amplitude of points that have been reached unphysically. If we assume that f is a function defined on Γ_h, then we have the following algorithm:

procedure `transport`(f)
  for $l = 0, \ldots, |\mathcal{V}_h| - 1$ **do**
    Let $\hat{\boldsymbol{x}}$ be the lth vertex in $\mathcal{V}_h$.
    if $\hat{\boldsymbol{x}}$ has a parent **then**
      Set $f(\hat{\boldsymbol{x}}) \leftarrow \sum_{i=0}^{n_{\text{par}}-1} \lambda_i f(\boldsymbol{x}_i)$.

Invoking `transport` as defined would transport $f \circ \boldsymbol{x}_0$ directly. An alternative would be to try to transport $\boldsymbol{x}_0$ and then compose it with f. However, we observe that developing an algorithm along the lines of `transport` to propagate $\boldsymbol{x}_0 = \boldsymbol{x}_0(\boldsymbol{x})$ throughout Ω is more involved. If S is curved, because of interpolation error, repeatedly iterating the linear map $(\boldsymbol{x}, \boldsymbol{\lambda}) \to \boldsymbol{x}_{\boldsymbol{\lambda}}$ will not result in an extension that remains in S. To ensure $\boldsymbol{x}_0(\boldsymbol{x}) \in S$, we would need to project $\boldsymbol{x}_{\boldsymbol{\lambda}}$ back onto S at each step, requiring the use of a spatial data structure to keep the time complexity manageable. After running "`transport`$(\boldsymbol{x}_0)$", in order to evaluate $f \circ \boldsymbol{x}_0$ we would need to convert each value $\boldsymbol{x}_0(\boldsymbol{x})$ into the same form as (4.37)—again, requiring a spatial data structure. On the other hand, taking convex combinations of values in a field like $\mathbb{R}$ or $\mathbb{C}$ is an inherently stable process. In fact, `transport` resembles the pyramid algorithms used to evaluate polynomial interpolants, such as De Casteljau's algorithm for evaluating Bernstein-Bézier polynomials, which are known to be stable [56]. For this reason, we prefer using `transport` to compute $f \circ \boldsymbol{x}_0$ directly.

We also note that using Bernstein-Bézier techniques allows us to generalize `transport` to handle higher-order interpolation easily. For example, if we knew the values of both

f and ∇f for each triangular face incident on Γ_h, and if we let $\mathsf{F}(\boldsymbol{\lambda}; \boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n_{\text{par}}-1})$ denote an approximation to f over $\Delta^{n_{\text{par}}-1}$ based on the available data (the constant value $f(\boldsymbol{x}_0)$ if $n_{\text{par}} = 1$, a cubic Hermite polynomial if $n_{\text{par}} = 2$, and the 9-parameter Bézier triangle if $n_{\text{par}} = 3$), then we can use the following algorithm to do higher-order accurate transport:

procedure `transport_bezier`(f, ∇f)
  for $l = 0, \ldots, |\mathcal{V}_h| - 1$ **do**
    Let $\hat{\boldsymbol{x}}$ be the lth vertex in $\mathcal{V}_h$.
    if $\hat{\boldsymbol{x}}$ has a parent **then**
      Assemble F from $f(\boldsymbol{x}_i)$ and $\nabla f(\boldsymbol{x}_i)$ for $i = 0, \ldots, n_{\text{par}} - 1$.
      Set $f(\hat{\boldsymbol{x}}) \leftarrow \mathsf{F}(\boldsymbol{\lambda}; \boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n_{\text{par}}-1})$.
      Set $\nabla f(\hat{\boldsymbol{x}}) \leftarrow \sum_{i=0}^{n_{\text{par}}-1} \lambda_i \nabla f(\boldsymbol{x}_i)$.

We did not develop this algorithm for this work, but it is a simple extension and necessary for incorporating higher-order amplitude corrections from the series GA ansatz of (1.7).

4.9 The point source amplitude

For a point source, since $c \equiv \text{const}$, $\alpha(x) = \alpha_0 \|x - \boldsymbol{x}_{\text{src}}\|^{-1}$ [112]. This gives us a simple way to approximate the amplitude for the initial wavefield due to the point source. Note that α is singular at $\boldsymbol{x}_{\text{src}}$. Since our goal is to synthesize an RIR, we can threshold α to 0 dB using a smooth taper to avoid artifically large amplitudes near $\boldsymbol{x}_{\text{src}}$. If c is varying, then specifying BCs for α near $\boldsymbol{x}_{\text{src}}$ is nontrivial and involves developing an asymptotic model of α near the point source [8, 97]. However, in room acoustics, since the speed of sound is typically a perturbation about a constant value due to small fluctuations in temperature, our simple model here should be reasonably accurate. There are other ways of modeling the amplitude near a point source that have been developed for use with high-order sweeping methods [107, 48].

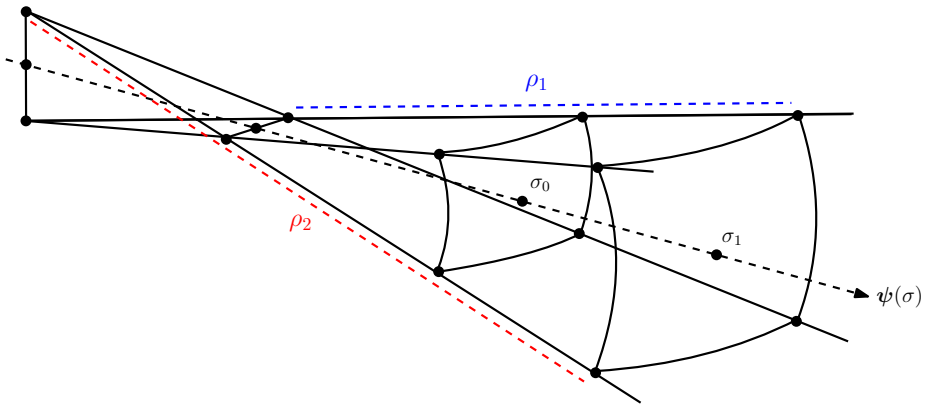


Figure 4.6: An astigmatic pencil of rays surrounding a fixed central ray, $\boldsymbol{\psi}(\sigma)$. The principal radii of curvature ρ_1 and ρ_2 are plotted for the parameter σ_1. The non-spherical wavefronts at parameters σ_0 and σ_1 are shown.

In Section 1.6, we develop reflection and diffraction BCs for the amplitude. For reflections, we assume that there is a reflection coefficient $R(\boldsymbol{x})$ defined on $\partial\Omega$ such that:

$$\alpha_{\text{out}} = R\alpha_{\text{in}} \tag{4.38}$$

where R is evaluated at the point of reflection. The coefficient R could be assumed to be frequency-dependent, although in this chapter we assume that each reflector has a constant, real value of $R \in (0, 1)$. For the diffracted amplitude, we use the simplified UTD coefficient for a flat wedge, given by (1.64) in Section 1.6.

4.10 The reflected and diffracted amplitudes

We can use `transport` to apply the BCs for the reflected and diffracted amplitudes throughout Ω. First, recall that the BCs for the reflected amplitude, taken from (1.52) and written out more explicitly, are:

$$\alpha_{\text{out}}(\boldsymbol{x}) = R(\boldsymbol{x}_0)\sqrt{\frac{\rho_1(\boldsymbol{x}_0)\rho_2(\boldsymbol{x}_0)}{\big[\rho_1(\boldsymbol{x}_0)+\sigma(\boldsymbol{x},\boldsymbol{x}_0)\big]\big[\rho_2(\boldsymbol{x}_0)+\sigma(\boldsymbol{x};\boldsymbol{x}_0)\big]}}\alpha_{\text{in}}(\boldsymbol{x}_0), \tag{4.39}$$

where $\boldsymbol{x}_0$ is the point of reflection, $\boldsymbol{x}$ is the observation point, ρ_1 and ρ_2 are the principal radii of curvature of the reflected wavefront at the point of reflection, and

σ is the arc length parameter of the ray leading from $\boldsymbol{x}_0$ to $\boldsymbol{x}$. Equation (4.39) is specialized to the case of $c \equiv 1$, but is general enough to handle different types of incident wavefronts. This is important: if a wave undergoes diffraction, the wavefront is no longer spherical. See Figure 4.6. Using this formula allows us to compute multiply diffracted and reflected arrivals.

We compute the compute the different radii of curvature in this section using (1.29). After marching the Hessian, we can recover the radius of curvature of the wavefront for the section defined by the wavefront's normal vector, and a vector tangent to wavefront denoted $\boldsymbol{v}$ from:

$$\rho_{\boldsymbol{v}}(\boldsymbol{x}_0) = \frac{1}{\boldsymbol{v}^\top \nabla^2 T(\boldsymbol{x}_0) \boldsymbol{v}}, \tag{4.40}$$

noting that the radii of curvature in this section are assumed to be positive. Since $\nabla^2 T \nabla T = 0$, since the principal directions $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are orthogonal to ∇T, the principal curvatures can be found by first computing eigenvalue decomposition of $\nabla^2 T$, giving:

$$\nabla^2 T(\boldsymbol{x}_0) = \kappa_1 \boldsymbol{v}_1 \otimes \boldsymbol{v}_1 + \kappa_2 \boldsymbol{v}_2 \otimes \boldsymbol{v}_2, \tag{4.41}$$

where $\kappa_1, \kappa_2, \boldsymbol{v}_1$, and $\boldsymbol{v}_2$ are the principal curvatures and directions, and then setting $\rho_i = 1/\kappa_i$ for $i = 1, 2$.

Using `transport` to compute the reflected amplitude. For our numerical experiments, we assume R to be constant over a reflecting surface, but for now assume that it is allowed to vary spatially. Additionally, note that:

$$\sigma(\boldsymbol{x}; \boldsymbol{x}_0) = \tau_{\text{out}}(\boldsymbol{x}) - \tau_{\text{in}}(\boldsymbol{x}_0), \tag{4.42}$$

noting that, since $c \equiv 1$, the eikonal (τ) and the arc length parameter (σ) are equal. Altogether, there are five different functions that need to be extended throughout Ω using `transport` to evaluate α_{out} using (4.39). This leads to the following algorithm:

procedure `evaluate_reflected_amplitude`(ρ_1, ρ_2, τ_{in}, α_{in}, R)
Solve eq. (1.4) to compute τ_{out}.
Use `transport` to extend $\rho_1, \rho_2, \tau_{\text{in}}, \alpha_{\text{in}}$, and R throughout Ω.
Compute α_{out} from (4.39).

To implement this algorithm, we could call `transport` five times, once for each field, or define a version of `transport` which traverses the dynamic programming plan once, transporting the full vector of data at each step. The time it takes to run `transport` is dwarfed by the time it takes to compute τ_{in}, rendering the choice unimportant. We do not dwell on this detail further.

Using `transport` to compute the diffracted amplitude. Computing the diffracted amplitude is a bit more involved. The diffracted amplitude is given by, from (1.55):

$$\alpha_{\text{out}}(\boldsymbol{x}) = D\big(\boldsymbol{x}_0\big)\sqrt{\frac{\rho_e(\boldsymbol{x}_0)}{\sigma(\boldsymbol{x};\boldsymbol{x}_0)\big[\rho_e(\boldsymbol{x}_0)+\sigma(\boldsymbol{x};\boldsymbol{x}_0)\big]}}\alpha_{\text{in}}(\boldsymbol{x}_0), \tag{4.43}$$

where:

$$D(\boldsymbol{x}_0) = D\big(L^{\text{in}}(\boldsymbol{x}_0), \varphi_{\text{in}}(\boldsymbol{x}_0), \varphi_{\text{out}}(\boldsymbol{x}_0), \beta_0(\boldsymbol{x}_0), n(\boldsymbol{x}_0)\big), \tag{4.44}$$

and where L^{in} depends on $\sigma(\boldsymbol{x};\boldsymbol{x}_0) = \tau_{\text{out}}(\boldsymbol{x}) - \tau_{\text{in}}(\boldsymbol{x}_0)$, $\rho_e(\boldsymbol{x}_0)$, $\rho_1(\boldsymbol{x}_0)$, $\rho_2(\boldsymbol{x}_0)$, and $\beta_0(\boldsymbol{x}_0)$. Here, the radius of curvature ρ_e is the radius of curvature of the incident wavefront in the plane of diffraction, which is defined the be the section spanned by $\nabla T(\boldsymbol{x}_0)$ and $\boldsymbol{t}_e$. To get at ρ_e, we compute:

$$\rho_e(\boldsymbol{x}_0) = \frac{1}{\boldsymbol{v}_e^\top \nabla^2 T(\boldsymbol{x}_0)\boldsymbol{v}_e}, \qquad \boldsymbol{v}_e = \frac{\big(\boldsymbol{I} - \nabla T(\boldsymbol{x}_0)\otimes\nabla T(\boldsymbol{x}_0)\big)\boldsymbol{t}_e}{\big\|\big(\boldsymbol{I} - \nabla T(\boldsymbol{x}_0)\otimes\nabla T(\boldsymbol{x}_0)\big)\boldsymbol{t}_e\big\|}. \tag{4.45}$$

The $\boldsymbol{t}_{\text{in}}$ and $\boldsymbol{t}_{\text{out}}$ fields. Altogether, computing α_{out} requires using `transport` to extend nine different fields: $\alpha_{\text{in}}, \tau_{\text{in}}, \varphi_{\text{in}}, \varphi_{\text{out}}, \rho_e, \rho_1, \rho_2, \beta_0$, and n. However, there is a caveat. The fields φ_{out} and β_0 both depend on the angle that the ray entering $\boldsymbol{x}_0$ makes with the ray leaving $\boldsymbol{x}_0$. This isn't well-defined at $\boldsymbol{x}_0$, so we can't transport

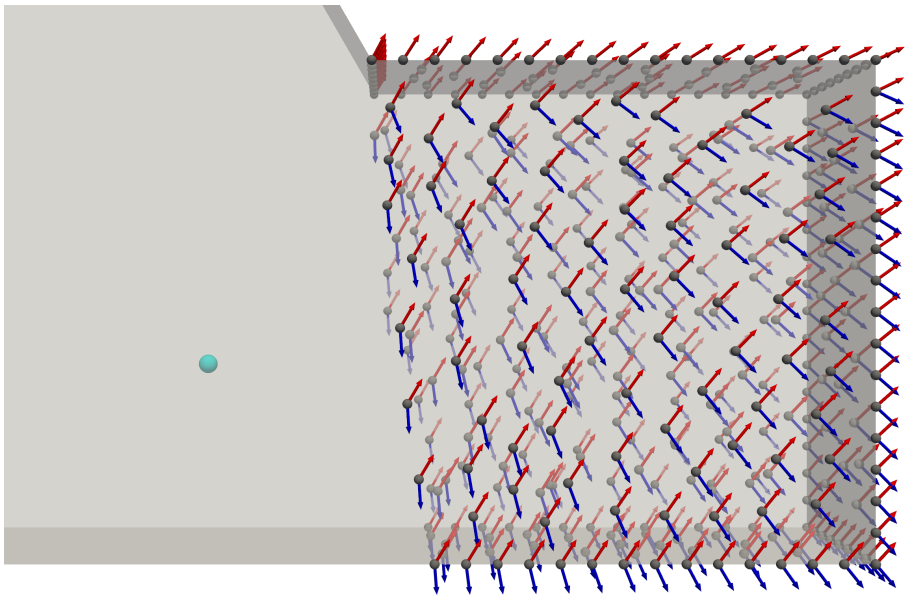


Figure 4.7: For the same reflected field show in Figure 4.5, we show the transported $\boldsymbol{t}_{\text{in}}$ and $\boldsymbol{t}_{\text{out}}$ vector fields, shown in red and blue, respectively. The cyan sphere denotes the original point source for the field reflected from the wall shown. The points where $\boldsymbol{t}_{\text{in}}$ and $\boldsymbol{t}_{\text{out}}$ do not satisfy the specular reflection condition will have their amplitude tapered, as discussed in Section 4.11.

these fields. To get around this problem, we define the $\boldsymbol{t}_{\text{in}}$ and $\boldsymbol{t}_{\text{out}}$ unit vector fields:

$$\boldsymbol{t}_{\text{in}}(\boldsymbol{x}) = \big(\nabla T_{\text{in}} \circ \boldsymbol{x}_0\big)(\boldsymbol{x}), \qquad \boldsymbol{t}_{\text{out}}(\boldsymbol{x}) = \big(\nabla T_{\text{out}} \circ \boldsymbol{x}_0\big)(\boldsymbol{x}). \tag{4.46}$$

See Figure 4.7. Here, $\boldsymbol{t}_{\text{in}}(\boldsymbol{x})f$ is the direction of the ray that reached $\boldsymbol{x}_0$ before carrying on to $\boldsymbol{x}$, and $\boldsymbol{t}_{\text{out}}$ is the direction which it left $\boldsymbol{x}_0$. Once we transport $\boldsymbol{t}_{\text{in}}$ and $\boldsymbol{t}_{\text{out}}$ throughout the domain, we can evaluate $\beta_0, \varphi_{\text{in}}$, and φ_{out} easily.

We compute each of these fields by using a variation on the transport algorithms described in Section 4.8. If $\boldsymbol{t} \in \mathbb{S}^2$ denotes the transported unit vector, and $\boldsymbol{t}_0, \boldsymbol{t}_1, \boldsymbol{t}_2 \in \mathbb{S}^2$ are the unit vectors on the parent triangle, a simple approach to adapting `transport` to vectors in $\mathbb{S}^2$ would be to set:

$$\boldsymbol{t} = \frac{\sum_{i=0}^{2} \lambda_i \boldsymbol{t}_i}{\left\| \sum_{i=0}^{2} \lambda_i \boldsymbol{t}_i \right\|}. \tag{4.47}$$

However, the error in this expression can be large. An alternative is to use spherical linear interpolation. Since each step of `transport` involves taking a weighted combination, we can consider generalizing the notion of taking a weighted combination

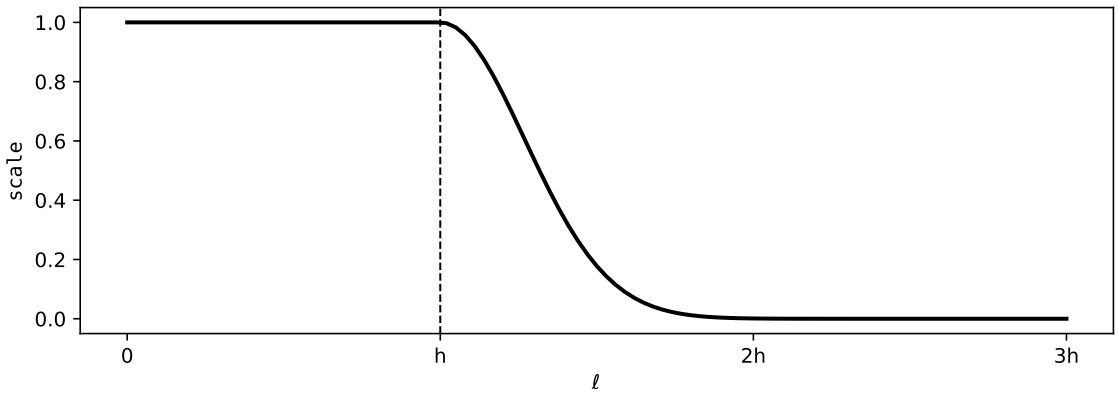


Figure 4.8: The basic scaling function used to taper the amplitude in the different types of shadow zone. Here, we set $\ell_{\text{thresh}} = h$ and $\alpha_{\min} = 10^{-3}$, so that $\texttt{scale}(2h) = 10^{-3}$. This is the same as a drop of 60 dB from a reference level. For larger values of ℓ, the taper continues to decay exponentially fast.

to unit vectors. A weighted combination of two unit vectors is well-defined, provided that they aren't antipodal—we set the combination to be the unit vector on the great circle connecting the two input vectors, and whose arc length to each input unit vector matches the desired weights [122]. This idea can be extended to an arbitrary number of unit vectors in $\mathbb{S}^{n-1}$ [26]. The idea is to find a unit vector which minimizes the energy:

$$E(\boldsymbol{t}) = \frac{1}{2}\sum_{i=0}^{n-1} \lambda_i d_g(\boldsymbol{t}, \boldsymbol{t}_i)^2, \tag{4.48}$$

where d_g is the geodesic distance. Optimization algorithms for computing $\boldsymbol{t}$ are fairly straightforward, and converge rapidly.

4.11 Tapering the amplitude of unphysical rays

Now that we have the necessary tools in place, we can discuss how to taper the amplitude of unphysical rays.

Tapering rays in the shadow zone. Since $c \equiv 1$, a we can check whether a ray lies in the shadow zone if:

$$\boldsymbol{t}_{\text{out}}(\boldsymbol{x})^\top \nabla \tau(\boldsymbol{x}) = 0 \tag{4.49}$$

holds, and is $\boldsymbol{x}$ is reachable from Γ_h by a ray contained entirely in the visible zone. For example, a ray may satisfy (4.49) but lie in the shadow zone—but the ray that reached that point will have diffracted first. To taper the amplitude of a ray in the shadow zone, we consider a sound to be inaudible if its amplitude reaches -60 dB from a reference amplitude of 1. This corresponds to an amplitude of $\alpha_{\min} = 10^{-3}$. We then define a threshold ℓ_{thresh} window function $\texttt{scale}(\ell)$ by:

$$\texttt{scale}(\ell; \ell_{\text{thresh}}) = \begin{cases} \exp\left(\log(\alpha_{\min})\left(\frac{\ell - \ell_{\text{thresh}}}{\ell_{\text{thresh}}}\right)^2\right), & \text{if } \ell \geq \ell_{\text{thresh}} \\ 1, & \text{otherwise.} \end{cases} \tag{4.50}$$

For $\ell > \ell_{\text{thresh}}$, $\texttt{scale}(\ell; \ell_{\text{thresh}})$ is a Gaussian window which tapers so that:

$$\texttt{scale}(2\ell_{\text{thresh}}; \ell_{\text{thresh}}) = \alpha_{\min}, \tag{4.51}$$

decreasing exponentially as ℓ increases. See Figure 4.8.

To taper the amplitude of unphysical rays in the shadow zone, we compute:

$$\ell_Z(\boldsymbol{x}) = \cos^{-1}\left(\boldsymbol{t}_{\text{out}}(\boldsymbol{x})^\top \nabla T(\boldsymbol{x})\right), \tag{4.52}$$

which has units of length, and set:

$$A_{\text{out}}(\boldsymbol{x}) \leftarrow \texttt{scale}(\ell_Z(\boldsymbol{x}); C_Z h) A_{\text{out}}(\boldsymbol{x}), \tag{4.53}$$

where C_Z is a small constant, typically $C_Z \approx 1$. Here, the subscript Z is used to denote the shadow zone. After tapering the amplitude using (4.53), we flood $\mathcal{V}_h$ using a breadth-first search starting from Γ_h to find the connected component of nodes with $A_{\text{out}} > \alpha_{\min}$ containing Γ_h. We zero the amplitude of all other nodes.

Tapering rays unphysical reflected and diffracted rays. For reflections, we compute:

$$\ell_R(\boldsymbol{x}) = \cos^{-1}\left(\boldsymbol{t}_{\text{in}}(\boldsymbol{x})^\top \left(\boldsymbol{I} - 2\boldsymbol{\nu}\boldsymbol{\nu}^\top\right) \boldsymbol{t}_{\text{out}}(\boldsymbol{x})\right). \tag{4.54}$$

This is the arc length between the $\boldsymbol{t}_{\text{out}}$ vector and the $\boldsymbol{t}_{\text{in}}$ vector after one of them has been reflected across the original reflecting plane. We then scale the reflected amplitude by:

$$A_{\text{out}}(\boldsymbol{x}) \leftarrow \texttt{scale}(\ell_Z(\boldsymbol{x}); C_Z h)\texttt{scale}(\ell_R(\boldsymbol{x}); C_R h) A_{\text{out}}(\boldsymbol{x}), \tag{4.55}$$

where C_R is another small constant chosen for the reflection scaling factor. For diffracted rays, we define:

$$\ell_D(\boldsymbol{x}) = \left|\cos^{-1}\left(\boldsymbol{t}_{\text{in}}(\boldsymbol{x})^\top \boldsymbol{t}_e\right) - \cos^{-1}\left(\boldsymbol{t}_{\text{out}}(\boldsymbol{x})^\top \boldsymbol{t}_e\right)\right| \tag{4.56}$$

and set:

$$A_{\text{out}}(\boldsymbol{x}) \leftarrow \texttt{scale}(\ell_Z(\boldsymbol{x}); C_Z h)\texttt{scale}(\ell_R(\boldsymbol{x}); C_D h) A_{\text{out}}(\boldsymbol{x}), \tag{4.57}$$

where C_D is a small constant for the diffraction scaling factor. Afterwards, for both reflected and diffracted fields, we flood $\mathcal{V}_h$ using a BFS starting from Γ_h to zero the amplitude of any nodes which have a spuriously large amplitude.

4.12 Numerical results

Numerical results for the first-arrival time

Since our goal is to approximate the Green's function for eq. (1.2), for our first arrival, we must solve eq. (1.4) with point source data. For the point source BCs $T(\boldsymbol{x}_{\text{src}}) = 0$, the eikonal asymptotically behaves like $\|x - \boldsymbol{x}_{\text{src}}\|$ near $\boldsymbol{x}_{\text{src}}$. This means that ∇T is singular at $\boldsymbol{x}_{\text{src}}$, which degrades the convergence of standard methods for solving the eikonal equation to $O(h \log \frac{1}{h})$ [143]. To recover the correct order of convergence, T must be initialized with sufficient accuracy in a constant radius ball around $\boldsymbol{x}_{\text{src}}$.

Another option is to solve the factored eikonal equation [52]. Factoring can be made to work with higher-order solvers [80]. The order of convergence also degrades when the wavefront diffracts around an edge or a vertex, where a rarefaction fan forms. In these cases, local factoring can be applied, but this has only been developed for corner singularities in 2D, but not yet been extended to 3D [106].

We use the `tetgen` program to construct a tetrahedron mesh, starting from a boundary representation stored as an OFF file [123]. Many other tools exist for generating tetrahedron meshes from boundary meshes [65, 55, 47]. We chose `tetgen` since we found it to be the simplest and easiest to use, it runs fast, and since the generated meshes are of reasonably high quality. In all of our experiments, we ran it using simple settings, just passing a uniform volume constraint on the size of the tetrahedrons, specifying the maximum quality setting, and otherwise leaving it alone. Our algorithms should not depend on the mesh generator used; and, despite the fact the meshes generated are Delaunay, do not appear to depend on this property.

Test problem: a point source in free space

We leave a proof of convergence of `jmm_free_space` for future work, but note that the proof for a varying speed of sound c should take much the same form as the proof for the 2D JMM [102]. To verify that the solver proposed in this section obtains the expected order of convergence, we consider the following test problem. We set $\Omega = [-1, 1]^3$, and discretize $\partial\Omega$ into a triangle mesh consisting of twelve faces (two per cube face). We then feed this through `tetgen` with a bound on the maximum tetrahedron volume. We set $c \equiv 1$, place a single point source at $\boldsymbol{x}_{\text{src}} = (0, 0, 0)$, and set $T(x) = \tau(x)$ if $\|x - \boldsymbol{x}_{\text{src}}\| \leq 0.1$. Note that $\tau(x) = \|x\|$ and $\nabla\tau(x) = (x - \boldsymbol{x}_{\text{src}})/\tau(x)$ for $x \neq \boldsymbol{x}_{\text{src}}$. We then solve eq. (1.4) on $\mathcal{V}_h$ using `jmm_free_space`, obtaining $O(h^2)$ accuracy in the relative ℓ_2 norm, as expected. We plot the error versus the average edge length of the mesh in Figure 4.9.

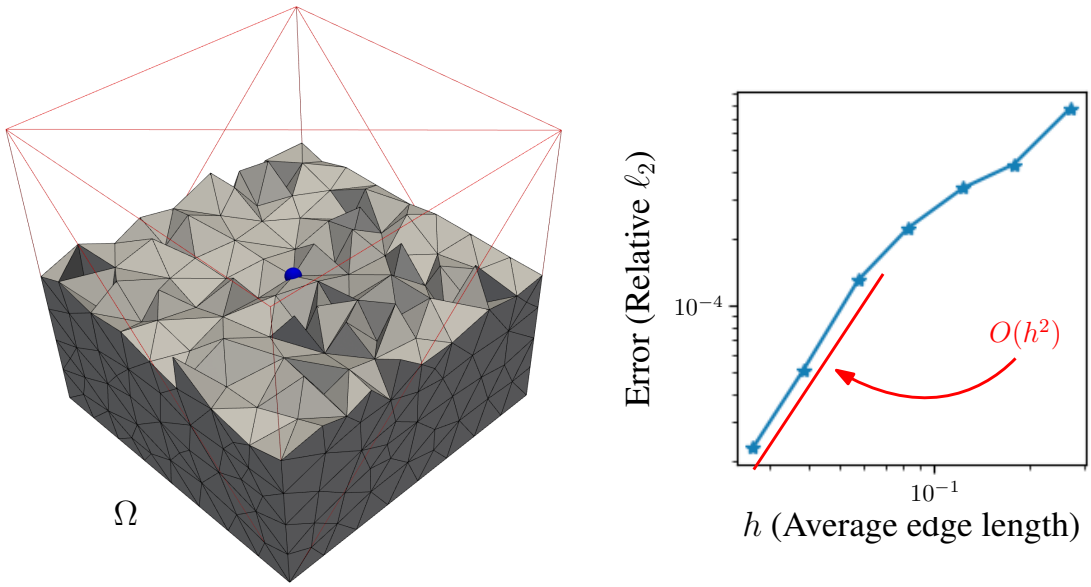


Figure 4.9: *Left*: a plot of a tetrahedron mesh discretizing the test problem in Section 4.12. The blue sphere indicates the location of the point source, and the mesh is cutaway at the xy-plane to give a sense of the mesh quality. *Right*: a plot showing the $O(h^2)$ order of convergence of `jmm_free_space`. For very small problems (top right), there are very few points in the interior of the mesh; once the points are dense enough, the rate of convergence obtains. All plots made using PyVista [130].

Test problem: a simple building domain

To compute reflected and diffracted fields in a more realistic setting, we conduct tests on a simple building domain. See Figures 4.10 and 4.11. We start with a surface mesh conforming to the boundary ($\partial\Omega$) of the interior of a building, and discretize is to a uniformly fine tetrahedron mesh using `tetgen`. The building fits inside the box $[-10, 10] \times [-10, 10] \times [0, 4.5]$, where we assume the dimensions are in meters. We discretize it into a mesh with 7,650 vertices and 31,581 tetrahedra. The mean, minimum, and maximum tetrahedron volumes are 4.53×10^{-2} m^3, 2.07×10^{-4} m^3, and 1.47×10^{-2} m^3, respectively. The mean, minimum, and maximum edge lengths are 7.55×10^{-1} m, 1.06×10^{-1} m, and 1.33 m, respectively. To be clear: this is a very coarse mesh. If were to discretize the surrounding bounding box using cells edge lengths equal to the average edge length of this mesh, we would get a regular grid in 3D with on the order of 4×10^3 vertices. So, we pay a penalty for using a tetrahedron

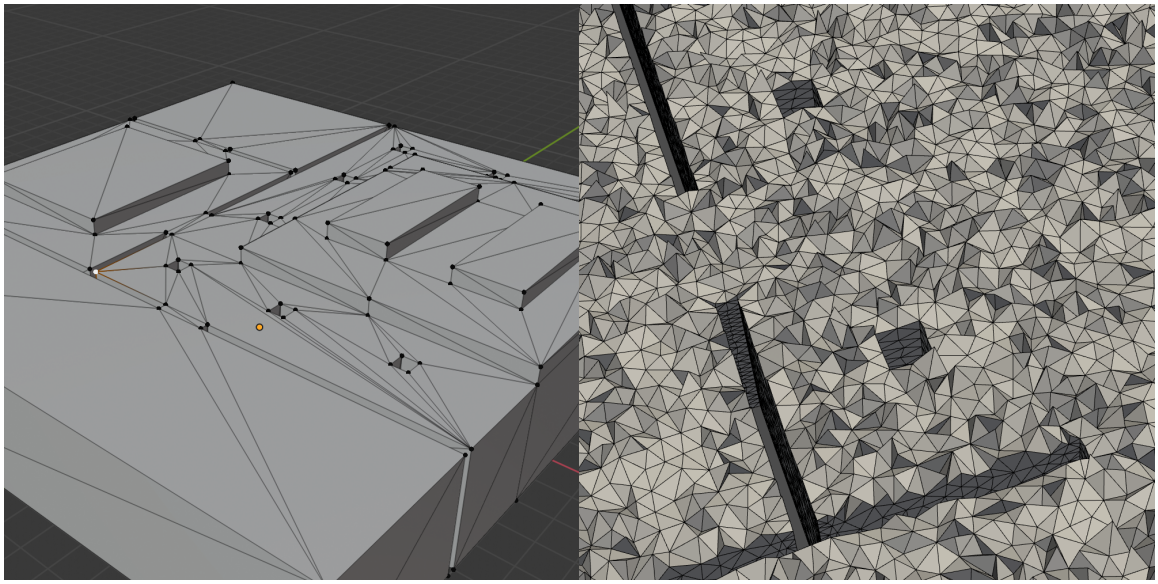


Figure 4.10: The test problem which is the focus of this chapter was modeled in Blender and meshed using TetGen. *Left*: A view of the building being modeled in Blender, showing the base triangle mesh used to discretize $\partial\Omega$. *Right*: A cutaway of the resulting tetrahedron mesh. As can be seen, the mesh is uniform, and does refine near corners or edges.

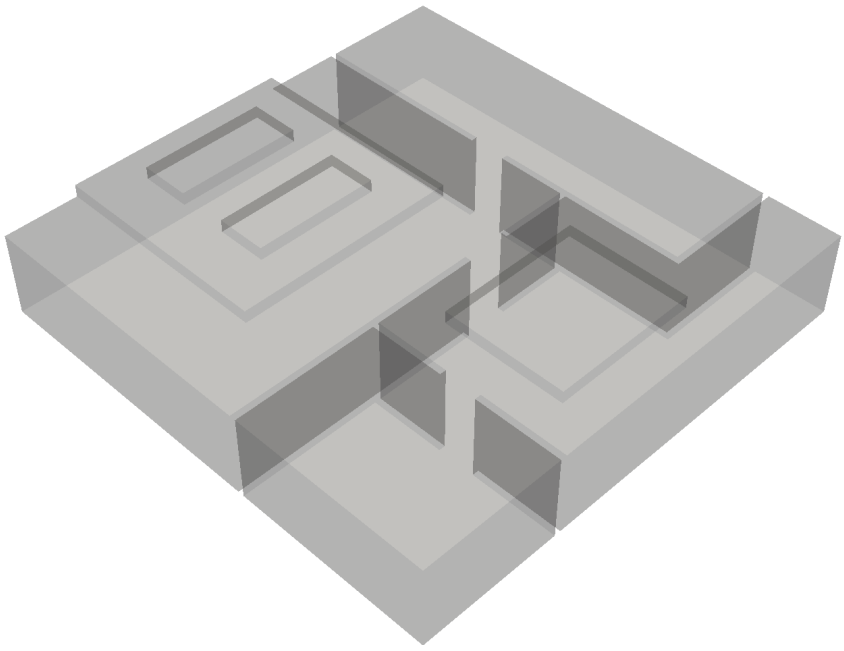


Figure 4.11: A full view of our building test problem. The building consists of four rooms connected by doorways, with different architectural features of modest complexity. Two of the rooms have recessed ceiling panels. The main room's recessed ceiling has skylight features.

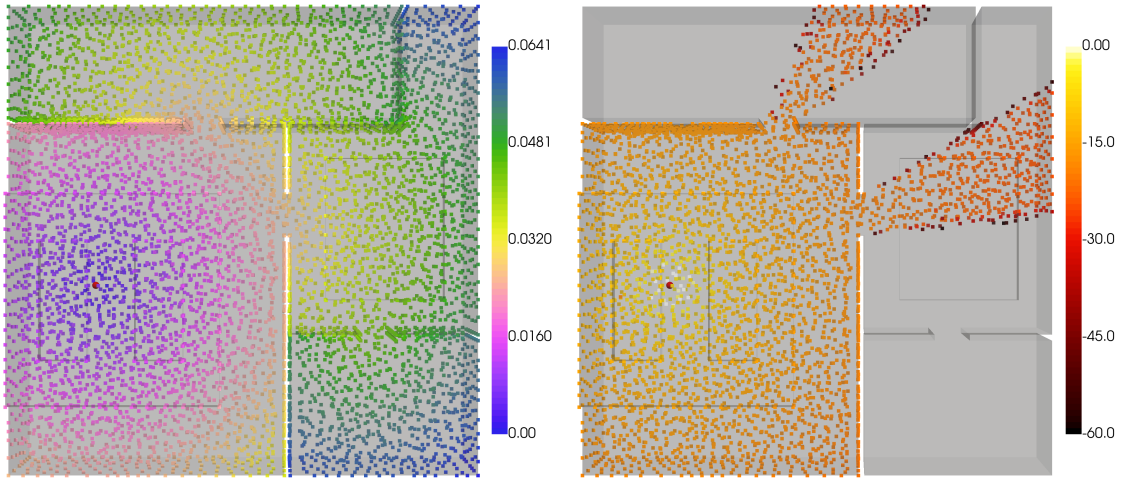


Figure 4.12: A view of the arrival time and amplitude for the point source indicated by the red sphere. *Left*: We show the complete time field to highlight how solving the eikonal equation as a PDE naturally incorporates diffraction into the solution. *Right*: The amplitude field, including tapering for vertices in the shadow zone, as described in Section 4.11. We remove vertices where the amplitude has fallen below -60 dB.

mesh, but not a large one.

To test out computing reflected and diffracted fields, we use the algorithms described in this chapter to solve an initial point source problem. This results in the time and amplitude fields shown in Figure 4.12. We then re-initialize our solver from different scattering features. See Figures 4.13 and 4.14.

Despite the extreme coarseness of the mesh, we can see that the boundaries separating the shadow zones, reflection zones, and diffraction zones are all captured reasonably well. The use of a smooth taper in Section 4.11 allows us to avoid discretizing or meshing these boundaries directly. For the reflected and diffracted fields, the tapering due to rays failing to satisfy the specular reflection or Keller cone conditions is easy to see.

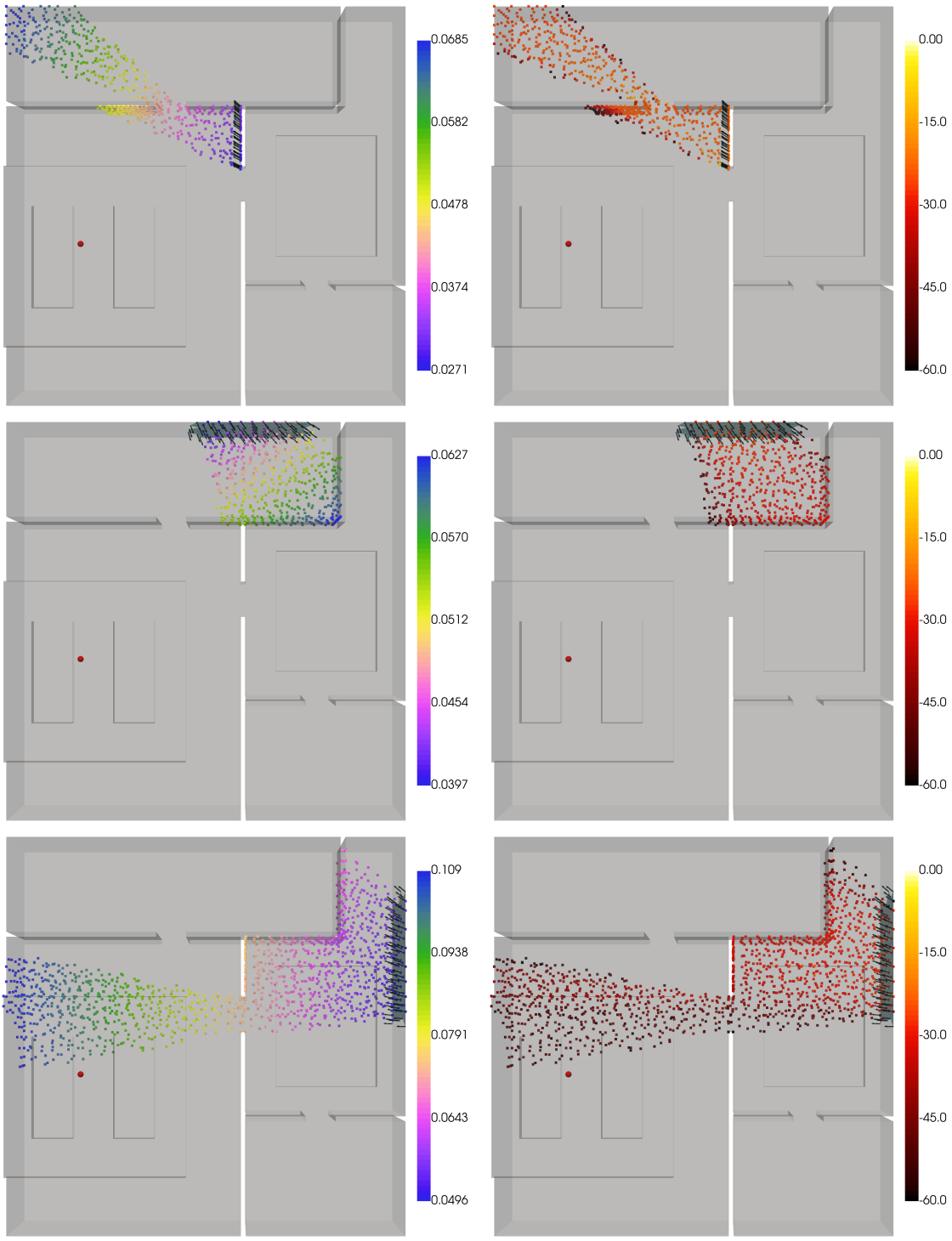


Figure 4.13: The reflected field for three different reflectors. Tapering due to the specular reflection condition being violated, and for vertices that lie in the shadow zone, can be observed. We plot the triangles where we specify reflection BCs, along with the values of ∇T_{out} (shown as arrows).

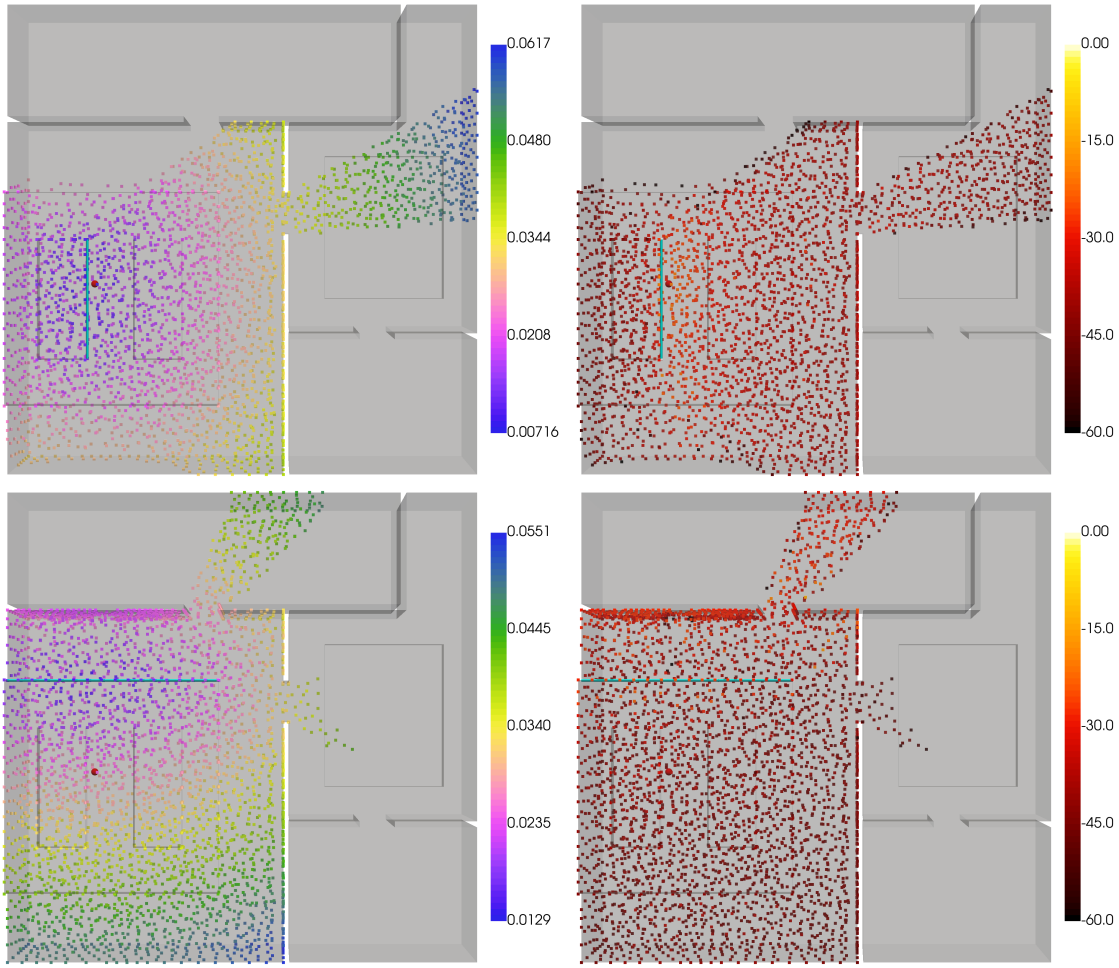


Figure 4.14: Scattering from two different diffractors. In each plot, we can see vertices which have their amplitude tapered because they lie outside of the diffraction zone (i.e., they do not lie on any Keller cone). *Top:* The nodes that lie above the diffracting edge lie outside the diffraction zone. *Right:* Nodes that lie to the right of diffractor are outside the diffraction zone. Additionally, tapering for nodes in the shadow zone takes effect, resulting in a fishtail (see Figure 4.15).

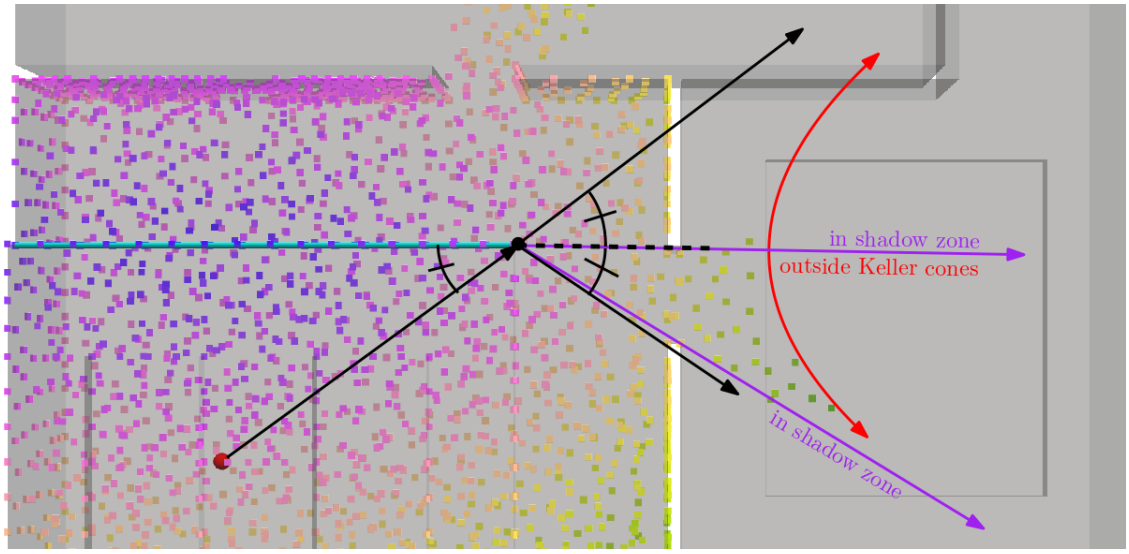


Figure 4.15: *Bottom:* A close-up of the fishtail. Rays the lie on individual Keller cones make the same angle arriving at the diffractor as they do leaving it. The rightmost point on the diffractor is a "terminal point", which emits a pencil of rays that behave as if they were originally generated by a point source. None of these rays lie on a Keller cone. Consequently, their amplitude will be tapered rapidly. At the same time, we can see that the shadow zone overlaps with the set of vertices which do not lie on a Keller cone. This results in a "fishtail" pattern.

Chapter 5

Conclusion

We have conducted an extensive study and development of semi-Lagrangian eikonal solvers in 2D and 3D on regular and unstructured meshes. This has resulted in the jet marching method (JMM), which marches the eikonal along with its first two derivatives, resulting in a high-order solver with a compact stencil that locally parametrizes rays. Furthermore, we have developed and explored an approach for modeling room acoustics termed "numerical geometric acoustics" that uses the 3D JMM for computing the travel times, their directions of arrival, and the amplitude on unstructured 3D meshes, and uses the geometric theory of diffraction to account for diffracted waves.

Our overarching goal is to develop a high-order JMM for solving the eikonal equation on an unstructured mesh discretizing a complicated, nonconvex domain, exemplified by the interior of a building or some exterior built environment. In Chapter 4, we present an eikonal JMM for the special case of $c \equiv 1$ which runs on a tetrahedron mesh, as well as algorithms for computing multiple arrivals. The idea here is to solve a tree of eikonal problems to approximate the multivalued solution of the high-frequency approximation of the Helmholtz equation. Each eikonal problem in the tree corresponds to a discrete reflected or diffracted wavefront, with the root of the tree corresponding to a generating point source. To compute the amplitude,

we additionally march the Hessian of the eikonal ($\nabla^2 T$), from which we extract a variety of curvatures needed to compute spreading factors. For diffracted wavefronts, we additionally need to evaluate the diffraction coefficient from the uniform theory of diffraction (UTD) in a semi-Lagrangian setting. To simplify the algorithm and avoid having to do surgery on discrete shadow and reflection boundaries, we taper the amplitude of the field using a Gaussian scaling factor. Each of these steps requires us to transport quantities downwind using precomputed dynamic programming plans recovered from the solution of the eikonal equation.

Adopting the semi-Lagrangian perspective enables higher-order solvers with more compact stencils. Explicitly parametrizing rays gives us helpful local information. We can integrate the ODEs governing the geometric spreading after computing a local update ray to indirectly march the amplitude. Along the same lines, we can propagate the Hessian integrating the matrix Riccati equation (eq. (3.47)) describing the evolution of the Hessian of the eikonal in the ray's normal plane, and combining the results. As well, since our local optimization problems explicitly parametrize the start of the ray as a convex combination of some number of "parent" nodes, we are able to recover a dynamic programming plan which we can use post-hoc to transport auxiliary quantities along the characteristics of the eikonal, but in a semi-Lagrangian setting.

5.1 Future work

In the remaining sections, we outline a few research projects which constitute natural next steps to continue the work in this

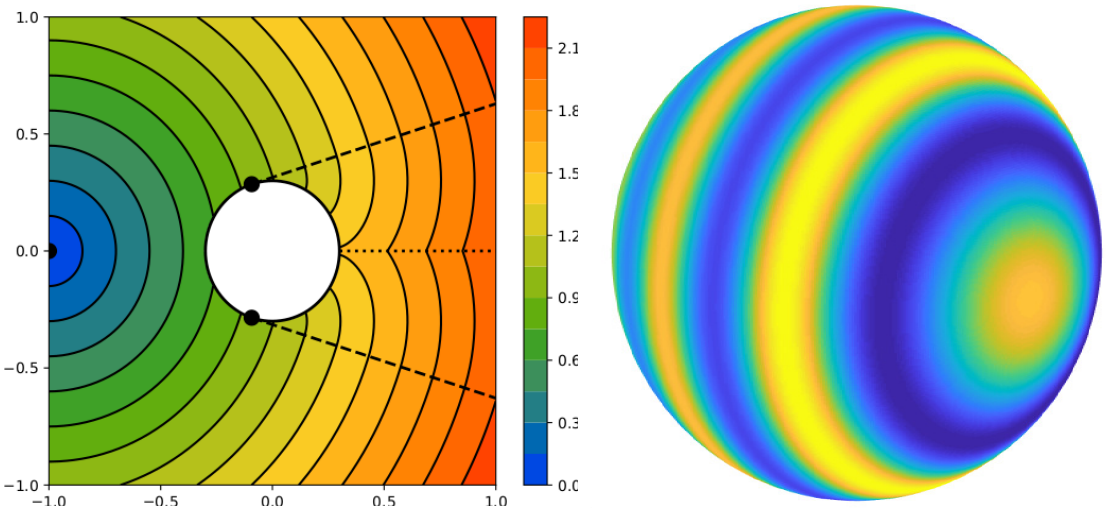


Figure 5.1: Scattering of a monochromatic wave from a sphere has an analytic solution given originally by Lord Rayleigh [129]. This problem requires us to model the rays shed tangentially into the shadow zone predicted by the uniform theory of diffraction [75], and would provide a useful test problem for an extension of our approach which handles curved obstacles in 3D. *Left*: The analytical eikonal for this problem, which is axisymmetric about the sphere. *Right*: The total field, including the contribution from the direct and scattered fields, plotted on the surface of a spherical scatterer for a particular frequency.

Numerical analysis of the jet marching method

We presented some preliminary results for the numerical analysis of the JMM in Chapter 3. The key task is analyzing the error amplification matrix described in Section 3.9. Generalizing these results in such a way that they are compatible with both a JMM on a regular grid in $\mathbb{R}^n$, as well as unstructured solver on a simplex mesh would be ideal. This would hopefully shed help us understand the consequences of solving the eikonal equation on an unstructured mesh more precisely.

Extending the 3D JMM

We plan on extending the 3D JMM presented in Chapter 4. There are several things that can be done:

- **Handling a varying speed of sound.** Our 3D tetrahedron mesh JMM was

developed for $c \equiv 1$ as a proof of concept. To handle c varying, we need to solve local raytracing problems analogous to those solved in Chapter 3. Solving these optimization problems in 3D is somewhat more complex, but still doable.

- **Using the Hessian to compute a higher-order interpolant.** Right now we compute an $O(h^3)$ interpolant using T and ∇T. As it stands, we also propagate the Hessian in order to compute the amplitude. We can also use the Hessian to approximate T using higher-order shape functions, such as the Bell triangle [37]. Once we extend the JMM handle a varying speed of sound, we can use the simplified paraxial ray theory presented in Section 1.5 to march $\nabla^2 T$ easily.

- **Incorporating a curved boundary.** To handle scattering from a curved obstacle, we will need to use conforming elements near the boundary, such as isoparametric elements [104]. There are analytically available formulae for scattering from spheres and ellipsoids, which can be used as ground truth test problems [129, 60, 2]. See Figure 5.1.

- **More complicated diffractional phenomena.** Incorporating a variable speed of sound and/or curved boundaries leads to more complicated diffraction effects. The author is unaware of whether UTD coefficients have been developed for a varying speed of sound. The primary references used for UTD for this dissertation both assume $c \equiv$ const [89, 85]. Additionally, scattering from a smooth, convex body results in rays which creep along the convex body and shed tangentially into the shadow zone. Such phenomena could also be incorporated.

Developing an end-to-end system for spatial audio

So far, we have developed a prototype algorithm for computing multiple arrivals in 3D (see Chapter 4). A task for the immediate future is collecting the results of Chapter 4 into a paper and submitting them. Additionally, we need to validate our numerical results for multiple arrivals against groundtruth test problems with analytically available solutions. This will allow us to characterize the performance of our solvers and make sure that they are behaving as expected. Following that, we will develop an algorithm either for extracting a fixed number of early arrivals or synthesizing parametric RIRs. We can then compare our method in terms of accuracy and runtime with existing approaches [110].

Following that, we would like to develop a system based on the results in Chapter 4 which can be used for spatial audio auralization (e.g., for use with a game engine). The main hurdle here is sensibly handling the proliferation of multiply reflected and diffracted fields. One approach would be to compute a fixed number of arrivals at each point. This can be expensive to do. An alternative approach would be to compute one or arrivals at each point and then using an energy-based method for the tail, such as acoustic radiosity [38, 133, 124].

Bibliography

[1] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[2] Ross Adelman, Nail A Gumerov, and Ramani Duraiswami. Semi-analytical computation of acoustic scattering by spheroids and disks. *The Journal of the Acoustical Society of America*, 136(6):EL405–EL410, 2014.

[3] Shahnawaz Ahmed, Stanley Bak, Joyce McLaughlin, and Daniel Renzi. A third order accurate fast marching method for the eikonal equation in two dimensions. *SIAM Journal on Scientific Computing*, 33(5):2402–2420, 2011.

[4] Andrei Alexandrescu. *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley, 2001.

[5] Jont B Allen and David A Berkley. Image method for efficiently simulating small-room acoustics. *The Journal of the Acoustical Society of America*, 65(4):943–950, 1979.

[6] Jörg Arndt. *Matters Computational: ideas, algorithms, source code*. Springer Science & Business Media, 2010.

[7] Uri M Ascher, Robert MM Mattheij, and Robert D Russell. *Numerical solution of boundary value problems for ordinary differential equations*, volume 13. Siam, 1994.

[8] Geraldo SS Ávila and Joseph B Keller. The high-frequency asymptotic field of a point source in an inhomogeneous medium. *Communications on Pure and Applied mathematics*, 16(4):363–381, 1963.

[9] V M Babich and N Y Kirpichnikova. *The boundary-layer method in diffraction problems*, volume 3. Springer, 1979.

[10] Ivo M Babuska and Stefan A Sauter. Is the pollution effect of the FEM avoidable for the Helmholtz equation considering high wave numbers? *SIAM Journal on numerical analysis*, 34(6):2392–2423, 1997.

[11] J-D Benamou, Songting Luo, and Hongkai Zhao. A compact upwind second order scheme for the eikonal equation. *Journal of Computational Mathematics*, pages 489–516, 2010.

[12] Jean-David Benamou. Big ray tracing: Multivalued travel time field computation using viscosity solutions of the eikonal equation. *Journal of Computational Physics*, 128(2):463–474, 1996.

[13] Jean-David Benamou. Multivalued solution and viscosity solutions of the eikonal equation. Tech report, 1997.

[14] Jean-David Benamou. *Direct computation of multi valued phase-space solutions for Hamilton-Jacobi equations*. PhD thesis, INRIA, 1998.

[15] Jean-David Benamou. An introduction to Eulerian geometrical optics (1992–2002). *Journal of scientific computing*, 19(1-3):63–93, 2003.

[16] Carl M Bender and Steven A Orszag. *Advanced mathematical methods for scientists and engineers I: Asymptotic methods and perturbation theory*. Springer Science & Business Media, 2013.

[17] Dmitri P Bertsekas. *Nonlinear programming*. Athena Scientific, 1999.

[18] Stefan Bilbao, Charlotte Desvages, Michele Ducceschi, Brian Hamilton, Reginald Harrison-Harsley, Alberto Torin, and Craig Webb. Physical modeling, algorithms, and sound synthesis: The NESS project. *Computer Music Journal*, 43(2-3):15–30, 2019.

[19] Stefan Bilbao and Brian Hamilton. Finite volume modeling of viscothermal losses and frequency-dependent boundaries in room acoustics simulations. In *Audio Engineering Society Conference: 60th International Conference: DREAMS (Dereverberation and Reverberation of Audio, Music, and Speech)*. Audio Engineering Society, 2016.

[20] Stefan Bilbao and Brian Hamilton. Passive volumetric time domain simulation for room acoustics applications. *The Journal of the Acoustical Society of America*, 145(4):2613–2624, 2019.

[21] Jeffrey Borish. Extension of the image model to arbitrary polyhedra. *The Journal of the Acoustical Society of America*, 75(6):1827–1836, 1984.

[22] Folkmar Bornemann and Christian Rasch. Finite-element discretization of static Hamilton-Jacobi equations based on a local variational principle. *Computing and Visualization in Science*, 9(2):57–69, 2006.

[23] Richard P Brent. *Algorithms for minimization without derivatives*. Courier Corporation, 2013.

[24] Alexander M Bronstein, Michael M Bronstein, and Ron Kimmel. *Numerical geometry of non-rigid shapes*. Springer Science & Business Media, 2008.

[25] Pere Brunet. Increasing the smoothness of bicubic spline surfaces. *Computer Aided Geometric Design*, 2(1-3):157–164, 1985.

[26] Samuel R Buss and Jay P Fillmore. Spherical averages and applications to spherical splines and interpolation. *ACM Transactions on Graphics (TOG)*, 20(2):95–126, 2001.

[27] Maria K. Cameron. https://youtu.be/Ze9AeDbaDVM, September 2020.

[28] Vlastislav Červenỳ, Mikhail M Popov, and Ivan Pšenčík. Computation of wave fields in inhomogeneous media—Gaussian beam approach. *Geophysical Journal International*, 70(1):109–128, 1982.

[29] A. Chacon and A. Vladimirsky. Fast two-scale methods for eikonal equations. *SIAM Journal on Scientific Computing*, 34(2):A547–A578, 2012.

[30] Adam Chacon and Alexander Vladimirsky. A parallel two-scale method for eikonal equations. *SIAM Journal on Scientific Computing*, 37(1):A156–A180, 2015.

[31] Chakravarty R Alla Chaitanya, Nikunj Raghuvanshi, Keith W Godin, Zechen Zhang, Derek Nowrouzezahrai, and John M Snyder. Directional sources and listeners in interactive sound propagation using reciprocal wave field coding. *ACM Transactions on Graphics (TOG)*, 39(4):44–1, 2020.

[32] Chakravarty R Alla Chaitanya, John M Snyder, Keith Godin, Derek Nowrouzezahrai, and Nikunj Raghuvanshi. Adaptive sampling for sound propagation. *IEEE transactions on visualization and computer graphics*, 25(5):1846–1854, 2019.

[33] Anish Chandak, Christian Lauterbach, Micah Taylor, Zhimin Ren, and Dinesh Manocha. AD-Frustum: Adaptive frustum tracing for interactive sound propagation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1707–1722, 2008.

[34] Siu-Wing Cheng, Tamal K Dey, and Jonathan Shewchuk. *Delaunay mesh generation*. CRC Press, 2012.

[35] David L Chopp. Some improvements of the fast marching method. *SIAM Journal on Scientific Computing*, 23(1):230–244, 2001.

[36] Philippe G Ciarlet. Interpolation error estimates for the reduced Hsieh-Clough-Tocher triangle. *Mathematics of Computation*, 32(142):335–344, 1978.

[37] Philippe G Ciarlet. *The finite element method for elliptic problems*. SIAM, 2002.

[38] Michael F Cohen and John R Wallace. *Radiosity and realistic image synthesis*. Elsevier, 2012.

[39] Michael G Crandall and Pierre-Louis Lions. Viscosity solutions of Hamilton-Jacobi equations. *Transactions of the American mathematical society*, 277(1):1–42, 1983.

[40] Daisy Dahiya and Maria Cameron. An ordered line integral method for computing the quasi-potential in the case of variable anisotropic diffusion. *arXiv preprint arXiv:1806.05321*, 2018.

[41] Daisy Dahiya and Maria Cameron. Ordered line integral methods for computing the quasi-potential. *Journal of Scientific Computing*, 75(3):1351–1384, 2018.

[42] Miles Detrixhe, Frédéric Gibou, and Chohong Min. A parallel fast sweeping method for the eikonal equation. *Journal of Computational Physics*, 237:46–55, 2013.

[43] Robert B Dial. Algorithm 360: shortest-path forest with topological ordering. *Communications of the ACM*, 12(11):632–633, 1969.

[44] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[45] Jean-Denis Durou, Maurizio Falcone, and Manuela Sagona. Numerical methods for shape-from-shading: A new survey with benchmarks. *Computer Vision and Image Understanding*, 109(1):22–43, 2008.

[46] Björn Engquist and Olof Runborg. Computational high frequency wave propagation. *Acta numerica*, 12:181–266, 2003.

[47] Andreas Fabri and Sylvain Pion. CGAL: The computational geometry algorithms library. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 538–539, 2009.

[48] Jun Fang, Jianliang Qian, Leonardo Zepeda-Núñez, and Hongkai Zhao. A hybrid approach to solve the high-frequency Helmholtz equation with source singularity in smooth heterogeneous media. *Journal of Computational Physics*, 371:261–279, 2018.

[49] Gerald Farin. Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design*, 3(2):83–127, 1986.

[50] Gerald Farin. *Curves and surfaces for computer-aided geometric design: a practical guide*. Elsevier, 2014.

[51] Michael S Floater. Chordal cubic spline interpolation is fourth-order accurate. *IMA Journal of Numerical Analysis*, 26(1):25–33, 2006.

[52] Sergey Fomel, Songting Luo, and Hongkai Zhao. Fast sweeping method for the factored eikonal equation. *Journal of Computational Physics*, 228(17):6440–6455, 2009.

[53] Sergey Fomel and James A Sethian. Fast-phase space computation of multiple arrivals. *Proceedings of the National Academy of Sciences*, 99(11):7329–7334, 2002.

[54] Thomas Funkhouser, Nicolas Tsingos, Ingrid Carlbom, Gary Elko, Mohan Sondhi, James E West, Gopal Pingali, Patrick Min, and Addy Ngan. A beam tracing method for interactive architectural acoustics. *The Journal of the acoustical society of America*, 115(2):739–756, 2004.

[55] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *International journal for numerical methods in engineering*, 79(11):1309–1331, 2009.

[56] Ron Goldman. *Pyramid algorithms: A dynamic programming approach to curves and surfaces for geometric modeling*. Elsevier, 2002.

[57] Ron Goldman. Curvature formulas for implicit curves and surfaces. *Computer Aided Geometric Design*, 22(7):632–658, 2005.

[58] Javier V Gómez, David Alvarez, Santiago Garrido, and Luis Moreno. Fast methods for eikonal equations: an experimental survey. *IEEE Access*, 2019.

[59] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. Siam, 2008.

[60] Nail A Gumerov and Ramani Duraiswami. *Fast multipole methods for the Helmholtz equation in three dimensions*. Elsevier, 2005.

[61] Nail A Gumerov and Ramani Duraiswami. Fast multipole accelerated boundary element methods for room acoustics. *arXiv preprint arXiv:2103.16073*, 2021.

[62] Hans Hagen and Guido Schulze. Automatic smoothing with geometric surface patches. *Computer Aided Geometric Design*, 4(3):231–235, 1987.

[63] Brian Hamilton and Stefan Bilbao. FDTD methods for 3-D room acoustics simulation with high-order accuracy in space and time. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(11):2112–2124, 2017.

[64] Paul S Heckbert and Pat Hanrahan. Beam tracing polygonal objects. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 119–127, 1984.

[65] Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. Tetrahedral meshing in the wild. *ACM Trans. Graph.*, 37(4):60–1, 2018.

[66] Ivo Ihrke, Gernot Ziegler, Art Tevs, Christian Theobalt, Marcus Magnor, and Hans-Peter Seidel. Eikonal rendering: Efficient light transport in refractive objects. *ACM Transactions on Graphics (TOG)*, 26(3):59, 2007.

[67] Won-Ki Jeong and Ross T Whitaker. A fast iterative method for eikonal equations. *SIAM Journal on Scientific Computing*, 30(5):2512–2534, 2008.

[68] Chiu-Yen Kao, Stanley Osher, and Jianliang Qian. Legendre-transform-based fast sweeping methods for static Hamilton-Jacobi equations on triangulated meshes. *Journal of Computational Physics*, 227(24):10209–10225, 2008.

[69] Joseph B Keller. Geometrical theory of diffraction. *JOSA*, 52(2):116–130, 1962.

[70] Seongjai Kim. An $O(N)$ level set method for eikonal equations. *SIAM journal on scientific computing*, 22(6):2178–2193, 2001.

[71] Seongjai Kim. 3-D eikonal solvers: First-arrival traveltimes. *Geophysics*, 67(4):1225–1231, 2002.

[72] Ron Kimmel and James A Sethian. Computing geodesic paths on manifolds. *Proceedings of the national academy of Sciences*, 95(15):8431–8435, 1998.

[73] Ron Kimmel and James A Sethian. Optimal algorithm for shape from shading and path planning. *Journal of Mathematical Imaging and Vision*, 14(3):237–244, 2001.

[74] Stephen Kirkup. The boundary element method in acoustics: a survey. *Applied Sciences*, 9(8):1642, 2019.

[75] Robert G Kouyoumjian and Prabhakar H Pathak. A uniform geometrical theory of diffraction for an edge in a perfectly conducting surface. *Proceedings of the IEEE*, 62(11):1448–1461, 1974.

[76] Michael Kremer, David Bommes, and Leif Kobbelt. OpenVolumeMesh–a versatile index-based data structure for 3d polytopal complexes. In *Proceedings of the 21st International Meshing Roundtable*, pages 531–548. Springer, 2013.

[77] Heinrich Kuttruff. *Room acoustics*. CRC Press, 2016.

[78] Ming-Jun Lai and Larry L Schumaker. *Spline functions on triangulations*. Cambridge University Press, 2007.

[79] Randall J LeVeque et al. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

[80] Songting Luo and Jianliang Qian. Factored singularities and high-order lax–friedrichs sweeping schemes for point-source traveltimes and amplitudes. *Journal of Computational Physics*, 230(12):4742–4755, 2011.

[81] Songting Luo and Jianliang Qian. Fast sweeping methods for factored anisotropic eikonal equations: multiplicative and additive factors. *Journal of Scientific Computing*, 52(2):360–382, 2012.

[82] Songting Luo, Jianliang Qian, and Robert Burridge. High-order factorization based high-order hybrid fast sweeping methods for point-source eikonal equations. *SIAM Journal on Numerical Analysis*, 52(1):23–44, 2014.

[83] Songting Luo and Hongkai Zhao. Convergence analysis of the fast sweeping method for static convex Hamilton-Jacobi equations. *Research in the Mathematical Sciences*, 3(1):35, 2016.

[84] Dejan Marković, Fabio Antonacci, Augusto Sarti, and Stefano Tubaro. 3D beam tracing based on visibility lookup for interactive acoustic modeling. *IEEE transactions on visualization and computer graphics*, 22(10):2262–2274, 2016.

[85] Derek A McNamara, Carl WI Pistorius, and JAG Malherbe. *Introduction to the uniform geometrical theory of diffraction*. Artech House Norwood, MA, 1990.

[86] Jean-Marie Mirebeau. Anisotropic fast-marching on cartesian grids using lattice basis reduction. *SIAM Journal on Numerical Analysis*, 52(4):1573–1599, 2014.

[87] Jean-Marie Mirebeau. Efficient fast marching with Finsler metrics. *Numerische mathematik*, 126(3):515–557, 2014.

[88] Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, 1987.

[89] Frédéric Molinet. *Acoustic high-frequency diffraction theory*. Momentum Press, 2011.

[90] Jean-Christophe Nave, Rodolfo Ruben Rosales, and Benjamin Seibold. A gradient-augmented level set method with an optimally local, coherent advection scheme. *Journal of Computational Physics*, 229(10):3802–3827, 2010.

[91] Richard D Neidinger. Introduction to automatic differentiation and matlab object-oriented programming. *SIAM review*, 52(3):545–563, 2010.

[92] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *ACM Sigplan Notices*, pages 89–100. ACM, 2007.

[93] Fred E Nicodemus. Directional reflectance and emissivity of an opaque surface. *Applied optics*, 4(7):767–775, 1965.

[94] J. Nocedal and S. Wright. *Numerical optimization*. Springer, 2006.

[95] Stanley Osher and Ronald Fedkiw. *Level set methods and dynamic implicit surfaces*, volume 153. Springer Science & Business Media, 2006.

[96] Mikhail Mikhailovich Popov. A new method of computation of wave fields using Gaussian beams. *Wave motion*, 4(1):85–97, 1982.

[97] Mikhail Mikhailovich Popov. *Ray theory and Gaussian beam method for geophysicists*. EDUFBA, 2002.

[98] MM Popov, Ivan Pšenčík, and V Červenỳ. Computation of ray amplitudes in inhomogeneous media with curved interfaces. *Studia Geophysica et Geodaetica*, 22(3):248–258, 1978.

[99] Alexander Mihai Popovici and James A Sethian. 3-D imaging using higher order fast marching traveltimes. *Geophysics*, 67(2):604–609, 2002.

[100] Samuel F. Potter. Dijkstra-like algorithms for high-frequency approximations of the Helmholtz equation. http://users.umiacs.umd.edu/ sfp/posters/ccsrd19.pdf, 2019.

[101] Samuel F. Potter and Maria K. Cameron. Ordered line integral methods for solving the eikonal equation. *Journal of Scientific Computing*, 81(3):2010–2050, 2019.

[102] Samuel F Potter and Maria K Cameron. Jet marching methods for solving the eikonal equation. *arXiv preprint arXiv:2009.05490*, 2020.

[103] Emmanuel Prados and Olivier Faugeras. Shape from shading. In *Handbook of mathematical models in computer vision*, pages 375–388. Springer, 2006.

[104] Paddy M Prenter et al. *Splines and variational methods*. Courier Corporation, 2008.

[105] Rok Prislan, Gregor Veble, and Daniel Svenšek. Ray-trace modeling of acoustic Green's function based on the semiclassical (eikonal) approximation. *The Journal of the Acoustical Society of America*, 140(4):2695–2702, 2016.

[106] Dongping Qi and Alexander Vladimirsky. Corner cases, singularities, and dynamic factoring. *Journal of Scientific Computing*, 79(3):1456–1476, 2019.

[107] Jianliang Qian, Lijun Yuan, Yuan Liu, Songting Luo, and Robert Burridge. Babich's expansion and high-order eulerian asymptotics for point-source Helmholtz equations. *Journal of Scientific Computing*, 67(3):883–908, 2016.

[108] Nikunj Raghuvanshi, Rahul Narain, and Ming C Lin. Efficient and accurate sound propagation using adaptive rectangular decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):789–801, 2009.

[109] Nikunj Raghuvanshi and John Snyder. Parametric wave field coding for precomputed sound propagation. *ACM Transactions on Graphics (TOG)*, 33(4):38, 2014.

[110] Nikunj Raghuvanshi and John Snyder. Parametric directional coding for precomputed sound propagation. *ACM Transactions on Graphics (TOG)*, 37(4):108, 2018.

[111] Nikunj Raghuvanshi, John Snyder, Ravish Mehra, Ming Lin, and Naga Govindaraju. Precomputed wave simulation for real-time sound propagation of dynamic sources in complex scenes. In *ACM Transactions on Graphics (TOG)*, volume 29, page 68. ACM, 2010.

[112] Lauri Savioja and U Peter Svensson. Overview of geometrical room acoustic modeling techniques. *The Journal of the Acoustical Society of America*, 138(2):708–730, 2015.

[113] Carl Schissler, Ravish Mehra, and Dinesh Manocha. High-order diffraction and diffuse reflections for interactive sound propagation in large environments. *ACM Transactions on Graphics (TOG)*, 33(4):39, 2014.

[114] Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-Wesley Professional, 2011.

[115] Benjamin Seibold, Jean-Christophe Nave, and Rodolfo Ruben Rosales. Jet schemes for advection problems. *arXiv preprint arXiv:1101.5374*, 2011.

[116] James A Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

[117] James A Sethian. Fast marching methods. *SIAM review*, 41(2):199–235, 1999.

[118] James A Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, volume 3. Cambridge University Press, 1999.

[119] James A Sethian and Alexander Vladimirsky. Fast methods for the Eikonal and related Hamilton–Jacobi equations on unstructured meshes. *Proceedings of the National Academy of Sciences*, 97(11):5699–5703, 2000.

[120] James A Sethian and Alexander Vladimirsky. Ordered upwind methods for static Hamilton-Jacobi equations: theory and algorithms. *SIAM Journal on Numerical Analysis*, 41(1):325–363, 2003.

[121] Georgy Evgen'evich Shilov and Richard A Silverman. *Linear Algebra*. Prentice-Hall, 1971.

[122] Ken Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, 1985.

[123] Hang Si. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software (TOMS)*, 41(2):1–36, 2015.

[124] Samuel Siltanen, Tapio Lokki, Sami Kiminki, and Lauri Savioja. The room acoustic rendering equation. *The Journal of the Acoustical Society of America*, 122(3):1624–1635, 2007.

[125] M Slotnick. Lessons in seismic computing. *Soc. Expl. Geophys*, 268, 1959.

[126] Gilbert W Stewart. *Afternotes Goes to Graduates School*. SIAM, 1998.

[127] J. Stoer and R. Bulirsch. *Introduction to numerical analysis*, volume 12. Springer Science & Business Media, 2013.

[128] Bjarne Stroustrup. *The C++ programming language*. Pearson Education, 2013.

[129] John W Strutt. On the acoustic shadow of a sphere. *Philos. Trans. R. Soc. London, Ser. A*, 203:8789, 1904.

[130] C Sullivan and Alexander Kaszynski. PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software*, 4(37):1450, 2019.

[131] Eran Treister and Eldad Haber. A fast marching algorithm for the factored eikonal equation. *Journal of Computational Physics*, 324:210–225, 2016.

[132] Yen-Hsi Richard Tsai, Li-Tien Cheng, Stanley Osher, and Hong-Kai Zhao. Fast sweeping algorithms for a class of Hamilton-Jacobi equations. *SIAM journal on numerical analysis*, 41(2):673–694, 2003.

[133] Nicolas Tsingos and Jean-Dominique Gascuel. A general model for the simulation of room acoustics based on hierarchical radiosity. In *SIGGRAPH'97 technical sketch*, 1997.

[134] John N Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, 1995.

[135] Jos Van Trier and William W Symes. Upwind finite-difference calculation of traveltimes. *Geophysics*, 56(6):812–821, 1991.

[136] Roelof Versteeg. The Marmousi experience: Velocity model determination on a synthetic complex data set. *The Leading Edge*, 13(9):927–936, 1994.

[137] John E Vidale. Finite-difference calculation of traveltimes in three dimensions. *Geophysics*, 55(5):521–526, 1990.

[138] Jui-Hsien Wang, Ante Qu, Timothy R Langlois, and Doug L James. Toward wave-based sound synthesis for computer animation. *ACM Trans. Graph.*, 37(4):109–1, 2018.

[139] Tao Xiong, Mengping Zhang, Yong-Tao Zhang, and Chi-Wang Shu. Fast sweeping fifth order WENO scheme for static Hamilton-Jacobi equations with accurate boundary treatment. *Journal of Scientific Computing*, 45(1-3):514–536, 2010.

[140] Shuo Yang, Samuel F Potter, and Maria K Cameron. Computing the quasipotential for nongradient SDEs in 3D. *Journal of Computational Physics*, 379:325–350, 2019.

[141] Liron Yatziv, Alberto Bartesaghi, and Guillermo Sapiro. $O(N)$ implementation of the fast marching algorithm. *Journal of computational physics*, 212(2):393–399, 2006.

[142] Yong-Tao Zhang, Hong-Kai Zhao, and Jianliang Qian. High order fast sweeping methods for static Hamilton-Jacobi equations. *Journal of Scientific Computing*, 29(1):25–56, 2006.

[143] Hongkai Zhao. A fast sweeping method for eikonal equations. *Mathematics of computation*, 74(250):603–627, 2005.

[144] Hongkai Zhao. Parallel implementations of the fast sweeping method. *Journal of Computational Mathematics*, pages 421–429, 2007.

[145] Dmitry N Zotkin, Ramani Duraiswami, and Larry S Davis. Rendering localized spatial audio in a virtual auditory space. *IEEE Transactions on multimedia*, 6(4):553–564, 2004.