Numerical Simulation of the Acoustical Propagation of Thunder

by
Jonathan S. Rood

A thesis submitted in
partial fulfillment of the
requirements for the degree
of
Doctor of Philosophy

Department of Mathematics
South Dakota State University

Abstract

Previous models for generating synthetic thunder lacked wave interaction due to lightning channel tortuosity. In this work, a two-dimensional CFD-type model based on the Navier-Stokes equations which includes the effects of shear viscosity, bulk viscosity, thermal conductivity, wind shear, refraction, and is capable of applying the effect of dispersion due to molecular relaxation of nitrogen and oxygen, is selected for utilization. The model is a set of six coupled equations, along with two equations of state to close the system. The model is augmented to be numerically solved using a hybrid scheme in space as well as the technique of adaptive mesh refinement (AMR). The hybrid scheme is based on the fusion of a fourth-order accurate dispersion-relation preserving (DRP) scheme with a weighted essentially non-oscillatory (WENO) scheme. Third-order and fifth-order accurate WENO schemes are explored for service. The WENO scheme requires more computations than the DRP scheme, but it is capable of propagating shock waves stably. The DRP scheme exhibits oscillations at discontinuities, but its computations are less complex than the WENO scheme. Therefore, the hybrid scheme must calculate a gradient of the domain and choose to use the WENO scheme where discontinuous waves may exist and the DRP scheme where continuous waves or no waves exist. A Runge-Kutta scheme is used to march the model through time. AMR is implemented through the Structured Adaptive Mesh Refinement Application Infrastructure (SAMRAI) developed by Lawrence Livermore National Laboratory. AMR allows the computer to focus computational effort on areas of the domain which will be the most beneficial. The result of this is the capability of the computer to handle a larger domain size than it could with a static mesh while maintaining comparable accuracy. To insert the complex geometry of a lightning channel source, a combination of mathematical functions is developed to efficiently insert the source and allow it to be sufficiently smooth to promote stability. The final result is a model that is capable of running on a large-scale parallel platform that can reproduce the acoustic signature of arbitrary tortuous lightning channel geometries including the effects of wave interaction over adequately-sized domains. However, lack of computational resources reduce the ability to achieve truly phenomenologically consistent results using the model.

Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

AMR	Adaptive Mesh Refinement
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy
DRP	Dispersion-Relation-Preserving
FFT	Fast Fourier Transform
GPU	Graphics Processing Unit
LLF	Local Lax-Friedrichs
MOL	Method of Lines
N-Wave	Nonlinear N-shaped Wave
PDF	Probability Density Function
PML	Perfectly Matched Layer
RK	Runge-Kutta
SAMRAI	Structured Adaptive Mesh Refinement Application Infrastructure
TVD	Total Variation Diminishing
WENO	Weighted Essentially Non-Oscillatory
WM-Wave	Wright-Medendorp Wave

Chapter 1

Introduction

1.1 Background

In the past, physical phenomena have been typically modeled by a computer through the use of illusionary type machination to achieve what would be considered reasonable results according merely to human perception. These models were composed this way largely due to computational limitations of computers. As computer performance has increased, so has the ability to model phenomena closer to their actual physical characteristics. The work in this dissertation focuses on a physically-based numerical model of the atmospheric phenomenon of shock waves produced by lightning which dissipate into acoustic waves known as thunder.

1.2 Significance of Study

The research in this dissertation is based upon the circumstance that a rigorous physically-based computational fluid dynamics (CFD) type model for the simulation and prediction of shock waves due to lightning strikes including tortuosity, and the resulting acoustic thunder had not yet been realized. The theme of this work is to contribute to the extension of the knowledge within the realm of this natural phenomenon and its prediction. A large aspect of that theme is the work of employing and advancing the techniques necessary in the attempt to overcome the computational complexity and intensity of such a task. Since a standard discretization of the domain into grid cells will be applied, the computational complexity arises as the computer is obligated to iterate over each cell in the grid(s) and perform calculations on them.

At each stage of increased computer performance and power, previous compromises which sacrificed features that were inherent to a problem, in order to lower its computational magnitude, can be revisited and possibly brought to fruition. Waves emitted from lightning strikes which included tortuosity, in previous models, lacked

interaction with one another, yet this effect is important to the acoustic signature of the thunder for particular lightning channel geometries. Therefore, this work brings tortuosity and wave interaction through a CFD-type model which is able to handle the nonlinear effects of thunder.

Aspects of the work presented here would be beneficial, yet not limited to: atmospheric scientists involved in thunder and lightning research, architects concerned with limiting thunderstorm noise in buildings, military scientists involved in researching effects of large explosions and the possible effects of military operations, foley artists involved in creating sound effects for movies, airplane manufacturers involved in limiting noise from jet engines, etc. Although the application of the model in this work is quite specific to simulating thunder and acoustics in air, the framework of the model itself is quite universal to other acoustic and wave-type problems.

1.3 Objectives

As stated previously, the purpose of this research is to extend and develop a model for simulating synthetic thunder in the computer in which waves emitted from the tortuous lightning channel are allowed to interact with one another as they propagate toward an observer. This interaction necessitates a CFD-type algorithm approach as the lightning channel geometry is structured according to a certain archetype, but essentially arbitrary. Therefore, the model does not have prior knowledge of its input and must be capable of handling complex source geometries. The drawbacks to CFD algorithms are their inherent computational magnitude. Therefore, an attempt is made to alleviate this drawback while simultaneously preserving the accuracy in which the model selected (described in Chapter 2) for this work is capable of achieving. However, sacrifices of features due to computational complexity will still be necessary; the most apparent of which will be a focus on modeling in two dimensions rather than three.

The random nature of lightning events and lack of absolute outcomes for a general lightning event enable this work to focus less on high accuracy simulations and more on generality, as well as the ability to vary the computational complexity when solving the model. However, in the interest of extending a useful model further, a modern and highly accurate model is selected, along with numerical methods for solving it which are able to be varied in accuracy (typically trading accuracy for computational efficiency). This general model is implemented while retaining its universal properties, and the methods its creator used in solving it are expanded upon, which are then welcome to be exploited in future nonlinear acoustics problems.

The technique of *adaptive mesh refinement* (AMR) is implemented using the Structured Adaptive Mesh Refinement Application Infrastructure (SAMRAI) software library developed by Lawrence Livermore National Laboratory. AMR allows the computer to focus computational effort and higher accuracy on parts of the model

in which it will benefit the most. It achieves this by managing a dynamic hierarchy of *patches* of grid cells that exist at grid levels of different resolutions and refining coarse grid cells into finer grid cells, or coarsening fine grid cells into coarser grid cells according to a programmer-defined mechanism for deciding which cells to adjust. Although this mechanism involves calculations across the grid that would be absent in a static grid approach, AMR is still typically much more efficient than if a static grid was used. This also means AMR allows the model to simulate domains larger in size than a standard static grid approach is capable of, assuming all areas of the domain do not need high resolution capability at all times during the simulation. Another advantage is its ability to increase or decrease accuracy in certain areas by refining cells on the grid to resolve features of a system with higher fidelity, or coarsen cells on the grid to ignore uninteresting features of a system. AMR will be described further in Chapter 4.

Another technique that is implemented is the use of a hybrid method for numerically solving the model. This method uses a high-accuracy method involving heavy computation (in this case, a *weighted essentially non-oscillatory* or WENO scheme), coupled with a high-accuracy method involving less computation (a *dispersion-relation preserving* or DRP scheme). The pitfall of the DRP scheme is that it does not propagate shock waves well. This is due to the shock waves being discontinuous and the DRP scheme exhibiting oscillations at discontinuities, whereas the WENO scheme can stably propagate discontinuous waves as well as continuous waves. Therefore, the WENO scheme will be assured use around the areas containing shock waves while the DRP scheme will be used over the rest of the domain where waves are continuous or do not exist. Again, this forces the computer to compute where these areas are, but the computations do not outweigh the efficiency gained in the hybrid approach as a whole.

An additional facet to this work is the insertion of complicated lightning channel geometries as sources for the model. When using AMR, the source should ideally have the ability to be described at arbitrary resolutions so that the finest level of the grid hierarchy can capture the source at the highest resolution immediately at insertion. An algorithm is devised so that this may be done efficiently and mathematically by using a fusion of Gaussian-distributed primitive functions. This allows for full control over the lightning channel geometry as well as the essential ability to construct the source in such a way that it is sufficiently smooth so as to not cause the methods of solving the model to become unstable.

The end result is a model that can convincingly predict the resulting thunder of lightning channel sources of arbitrary geometries over extended distances in two dimensions, while including the effects of wave interaction, nonlinearity, modified classical absorption, refraction, wind shear, as well as dispersion due to molecular relaxation.

1.4 Literature Review of Previous Work

Although acoustic modeling is a frequent area of study and application, the production of synthetic thunder specifically is fairly uncommon. However, research related to this area has taken place. In 1968 Wright and Medendorp[1] developed a technique to model finite-length sparks in air through a superposition of spherically propagating N-waves (N-shaped pressure waves which will be discussed further in Section 1.6.2; also see Figure 1.1) emitted from a line source. Their work was used extensively

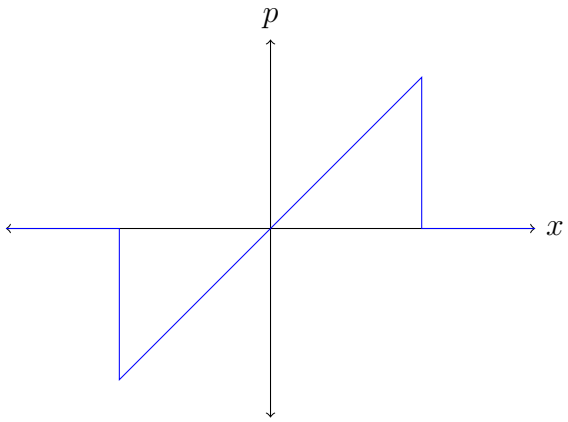

Figure 1.1: A pressure profile of a theoretically-shaped N-wave traveling to the right.

in later models, and modeling waves using their techniques became known as using WM-waves. They found that at certain angles of a straight-line channel emitting an N-wave, an observer would experience differently shaped waveforms. If an observer was directly perpendicular to the source, that observer would experience a standard N-wave. As the observer moves at an angle θ up or down, the ends of the waveform become parabolic arcs. Moving further, to the critical angle where $\sin\theta = \psi$, the waveform becomes a conjoining of two parabolas. Finally, moving past the critical angle results in the waveform becoming two parabolas with a zero-value line segment between them. The equations that are used to construct these WM-waves are listed in Table 1.1. A visual interpretation of WM-wave theory can be seen in Figure 1.2.

Plooster[2] and Bass[3], in 1971 and 1980, respectively, modeled the lightning channel as a purely cylindrical source. If one were to listen to the thunder provided by a large cylindrical source, only a single resulting "clap" would be heard. Approximating the lightning channel as a purely cylindrical source was necessary at that time due to the complexity involved with a loss of axisymmetry that would be introduced into their nonlinear models otherwise. Interestingly, there were discrepancies between their results in regards to the energy output per unit length of a lightning segment that were two orders of magnitude in difference[4]. In 1982 Few[5] modeled

Case of $\sin\theta \geq \psi$:

$$P = \begin{cases} 0 & \tau < -\psi - \sin\theta \\ (-B/\sin\theta)[(\tau + \sin\theta)^2 - \psi^2] & -\psi - \sin\theta < \tau < \psi - \sin\theta \\ 0 & \psi - \sin\theta < \tau < -\psi + \sin\theta \\ (B/\sin\theta)[(\tau - \sin\theta)^2 - \psi^2] & -\psi + \sin\theta < \tau < \psi + \sin\theta \\ 0 & \psi + \sin\theta < \tau \end{cases}$$

Case of $\sin\theta < \psi$:

$$P = \begin{cases} 0 & \tau < -\psi - \sin\theta \\ (-B/\sin\theta)[(\tau + \sin\theta)^2 - \psi^2] & -\psi - \sin\theta < \tau < -\psi + \sin\theta \\ -4B\tau & -\psi + \sin\theta < \tau < \psi - \sin\theta \\ (B/\sin\theta)[(\tau - \sin\theta)^2 - \psi^2] & \psi - \sin\theta < \tau < \psi + \sin\theta \\ 0 & \psi + \sin\theta < \tau \end{cases}$$

$\tau = \frac{ct-r}{l}, \quad \psi = \frac{cT}{l}, \quad B = \frac{Al^2}{2rcT}$

τ is the retarded time

ψ is the critical angle

B is the amplitude coefficient

θ is the angle from the segment normal to the observer

A is an arbitrary scaling factor

T is the duration of the wave

l is the length of the segment

c is the speed of sound

r is the distance to the observer

Table 1.1: Equations for WM-wave theory.

the lightning channel as a straight line cylindrical source at first and then as spherical after a certain distance of propagation he noted as the "relaxation radius". His claim is that this idea is appropriate due to the highly tortuous nature of lightning. Although these models that approximate the lightning channel as a cylindrical source helped qualitatively in the study of lightning and thunder, they would not be good at generating convincing audible thunder due to such a major simplifying assumption of lightning channel geometry.

In 1982 Ribner and Roy[6] knew that the tortuous nature of the lightning channel should affect the thunder's acoustic signature, as previous researchers have stated is the case, and devised a model to account for such tortuous channels. Their quasilinear model was based on WM-waves. Ribner and Roy used this idea to model the lightning channel in three dimensions as a chain of 3-meter segments in a tortuous fashion

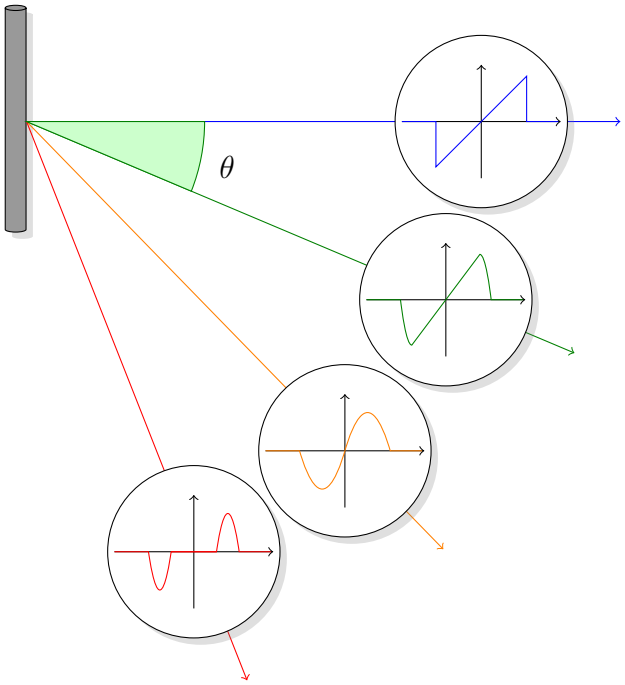

Figure 1.2: A visual representation of WM-wave theory displaying how the observer perceives the N-waves emitted from a line segment, shown according to their angle θ from the normal of a line segment.

without branching, which emitted N-waves from each segment in the channel. Due to the varying times in which the waves would reach the observer from different heights of the lightning channel, a convolution of waves was constructed from all the segments at their different angles (according to WM-wave theory) that the observer would then experience. A visual representation of this can be seen in Figure 1.3. They described the work as "*a mapping of the shape of lightning into the sound of thunder.*"[6] While this model can compute the synthetic thunder quickly, it lacks the ability for waves to interact with one another over time.

In 1991 Sparrow and Raspet[7] developed a two-dimensional model for nonlinear acoustics which allowed them to model complicated wave geometry similar to the model that will be used in this work and applied it to spark pulses. The model was expanded upon later by Wochner[8], which will be detailed in Chapter 2. Wochner applied the model to research in jet noise.

In 1999 Glassner[9] used the WM-wave model principle and added a visualization for the lightning channel as well. In Glassner's work it was also possible for the user of his model to start with the thunder by designating desired parameters in a broad manner, such as adjusting levels for "rumble", "roll" and "clap", and then a lightning channel geometry could be generated to adhere to those qualities. His model accounted for the effects of the atmosphere such as refraction, reflection, dispersion

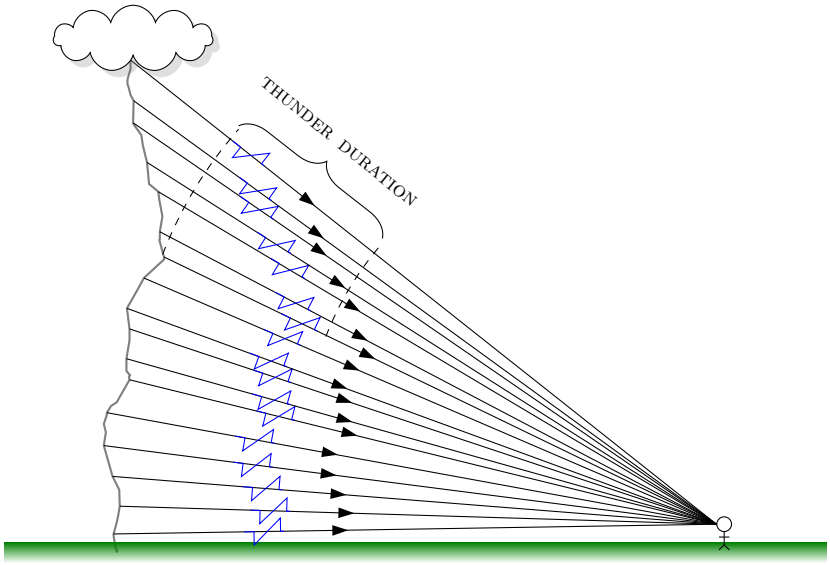

Figure 1.3: A visualization of Ribner's model with an observer to the right of the lightning channel. The duration of thunder can be seen as the length from the beginning of the first wave to the end of the last wave to reach the observer.

and wind shear by randomizing the flight time of the different waves from each segment. In 2007 Matsuyama *et al.*[10][11] based their work on Glassner's model and again used the same WM-wave principles while focusing on realtime audible generation of synthetic thunder and visualization of synthetic lightning that users could interact with on a touch screen monitor to enact lightning strikes. They chose to generate the lightning channels by modeling the electric field which involved solving a Laplace equation by using the conjugate gradient method which was implemented on the GPU.

The advantage of the models using WM-waves is that they allow for undemanding computations. The algorithm only needs to traverse each line segment to calculate its angle to the observer and its distance from the observer and then integrate all the WM-waves as the observer would encounter them in time. In this sense, very large domains can be modeled easily, but the approach is not physically accurate in a sense that the interaction between waves cannot be accounted for on their journey toward the observer.

1.5 Explored Models

Although WM-waves appear to be the preferred model to use for generating synthetic thunder, there are several models that exist that could be adapted to the problem in certain ways. Given below is a short list of models that were explored for consider-

ation in this work. Some of their notable advantages and disadvantages are listed. The most prevalent disadvantage to many of the models is their assumption that wave propagation will be primarily in a single direction, which is undesirable in this research.

1.5.1 Eikonal Equation

The eikonal equation is able to approximate some acoustic solutions and can even satisfy the wave equation in certain cases when the frequency is very high[12]. It is generally used in the ray acoustics approach which is very close to the approach taken by the majority of previous work mentioned implementing WM-waves. This means that if the model is to account for tortuousity, it will likely be approximating it with a superposition of acoustic rays just as Ribner and Roy[6] did. The problem of simulating thunder is also only concerned about capturing low frequencies, where the eikonal equation approximates high frequencies better. It could probably be used with success for a realtime type method for crudely approximating this problem.

1.5.2 Generalized Burgers Equation

The Burgers equation is a popular method for handing nonlinear acoustic situations. It exists in a wide range of forms. Although the classical (sometimes called lossless) Burgers equation is exactly solvable, the generalized form for the Burgers equation is not[13]. One form of generalization which would be necessary for the work here would be a form including dissipation. All the effects necessary for nonlinear acoustics in air have been built into the equation at some time during its existence. Although the generalized Burgers equation will not be used in the model in this work, its main benefit is its reasonable simplicity, and should be considered if simplicity is of higher concern.

1.5.3 KZK Equation

Put simply, the KZK equation is based on the Burgers equation but it adds in the capability of diffraction which occurs when a wave encounters an obstacle. Accounting for diffraction can also increase accuracy. If a generalized Burgers equation model were to be chosen, a KZK equation should be considered as well.

1.5.4 Weak-Shock Theory

Weak shock theory has been described as "*a very effective alternative to the Burgers equation for cases in which dissipation is primarily due to shocks in the traveling wave.*"[13] Since the Burgers equation is a suitable candidate for modeling the problem in this work, and weak-shock theory can be generalized beyond plane waves, it is

a candidate as well. However, there is no explicit built-in absorption mechanism. Absorption is included when using the Rankine-Hugoniot jump conditions. In principle, weak-shock theory is a combination of a lossless Burgers equation used to model the continuous sections of the waveform between shocks, along with an approximation of the Rankine-Hugoniot shock relations to calculate the position and amplitude of each shock. Weak-shock theory loses its accuracy once the shocks disperse and no longer dominate dissipation effects[13]. This is not ideal for the long-range propagation that is necessary to consider in this problem.

1.5.5 Pestorius Algorithm

The Pestorius algorithm is based on weak-shock theory and the Burgers equation but overcomes the shortfall of weak-shock theory by periodically accounting for the effects of ordinary dissipation to make it more accurate after shocks disperse. Since this algorithm has the Burgers equation involved, it should be capable of handling this problem. However, this algorithm was somewhat eclipsed by the Anderson algorithm which took out weak-shock theory and used thermoviscous losses instead[8]. These algorithms could possibly accommodate the problem of simulating thunder, but lack the interaction capabilities of CFD-type acoustic models.

1.5.6 Nonlinear Progressive Wave Equation

All the effects needed for propagation of thunder have been built into the nonlinear progressive wave equation (NPE) by Leissing[14] who used it to successfully model point source explosions in the atmosphere by coupling it with a linear model at a point where the nonlinear effects have dispersed. A graphical depiction can be seen in Figure 1.4. This coupling of a nonlinear and linear model make it well suited for this problem, as well as its generality. However, a fundamental assumption in the NPE is that the propagation of waves must be contained within a "moving window". It is possible the propagated waves from a tortuous lightning channel would not be hindered in a moving window and this window would help computational efficiency. However, this model is not as generalized as would be best for this problem, due to that assumption.

1.5.7 Wochner's Algorithm

Wochner's[8] work is somewhat of an extension of Sparrow and Raspet[7] as well as Pierce[15] in which he included the effects of molecular relaxation. He has designed a very generalized model capable of handling the necessary requirements for the complexity of this problem, and it would be advantageous to expand upon by increasing its computational efficiency and ability to manage large distances of propagation.

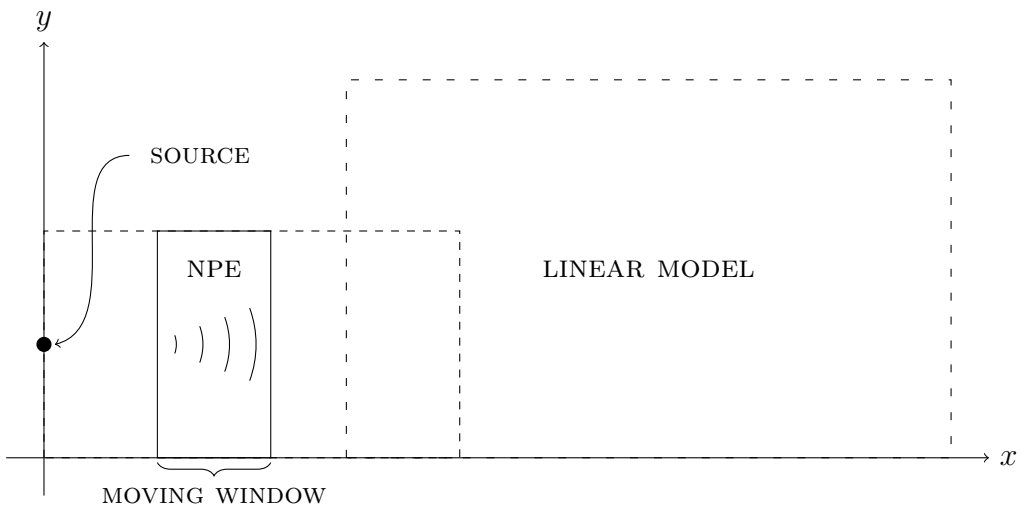

Figure 1.4: The coupling of a nonlinear model with a linear model by the use of a moving window.

The computational fluid dynamics principles that it is based on will give the tortuous lightning channel source, wave interaction capability that the other models listed would not be able to predict. However, it has one large drawback; in its initial implementation by Wochner, it is very burdensome to a computer both computationally and storage-wise. It is for this reason that this work will put together techniques in order to alleviate this shortcoming and still be able to take advantage of its powerful universality.

1.6 Model Requirements

In this section the requirements for a model which can adequately describe the problem of focus will be discussed. This will occur through a detailing of the process of lightning and thunder, linear and nonlinear acoustics in air, effects in air which attenuate waves, and the discretization of a requisite domain size for the problem. Wochner's model will meet all of these requirements, but fail on the ability to efficiently handle a large number of grid cells.

1.6.1 Characteristics of Lightning

The physical processes of lightning were considered a mystery from the dawn of civilization until about 1888 when M. Hirn proposed the theory that is accepted today. For a summarized historical background of theories regarding lightning, the reader is referred to Roy[16].

Thunder is a direct result of lightning. Therefore, to model thunder effectively, the traits and processes of lightning should first be overt. There are several different categories that various lightning events can be divided into. This work will be concerned with the cloud-to-ground type lightning event in which the lightning channel extends from a cloud and connects with the ground.

Typical cloud-to-ground lightning occurs in the following fashion. A negative charge first accumulates in a cloud. Once this negative charge exceeds the ionization potential of the air, the molecules in the lightning channel path break down, which lowers its resistance and allows the negative charge to travel through the channel. This initial flow of charge is known as the *stepped leader*. The stepped leader progresses by forking its way down from the cloud by building new negative charges, breaking down molecules, and then thrusting ahead. Building a charge takes about 50μs to occur and about 1μs for the charge to burst forward. This takes a total of about 20 ms for the stepped leader to reach the ground. The stepped leader is compelled to find its way to the ground by a difference of potential averaging 300 million volts[9]. Although it is difficult to see the stepped leader in realtime, video of lightning filmed with very high speed cameras record enough frames for humans to play back in slow motion and observe them. A diagram of a cloud-to-ground lightning event can be see in Figure 1.5.

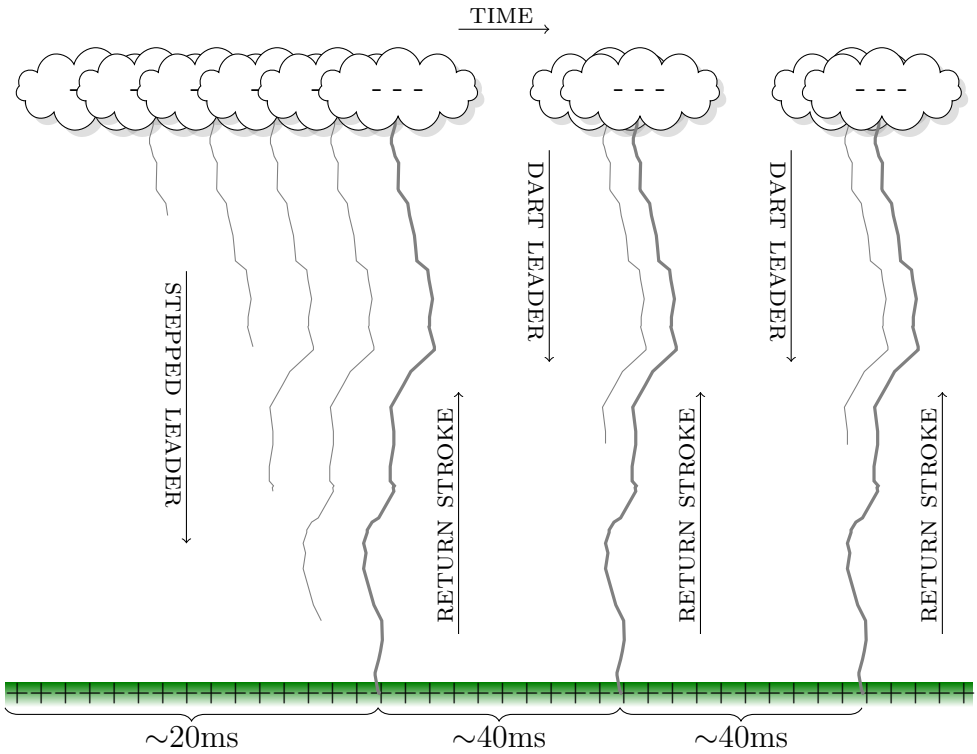

Figure 1.5: Visual representation of the timeline for an example lightning event with three return strokes.

As the stepped leader approaches the ground, positively charged *streamers* move upward from the ground to meet it. Once the channel connects with a streamer, the negative charge from the cloud careens toward the ground in an attempt to neutralize the potential between them. This is called the *return stroke*. At this point the channel emits a bright flash due to a heating to plasma temperatures of about 24,000 K. The heating of the channel to such temperatures is the genesis of thunder which will be discussed further in Section 1.6.2. This process can happen repeatedly with successive re-strikes of return strokes initiated by *dart leaders*. Succesive dart leader initiated return strokes occur in a very short amount of time (generally less than 120 ms between each electrical surge) due to charges from other parts of the cloud quickly finding their way to the low-resistance channel, and beginning the creation of a new dart leader. However, now the lowest-resistance path has been created and it will be used subsequently for multiple dart leaders until all charges in the cloud are neutralized. These multiple surges will be considered one lightning event. The number of times these dart leaders generally occur in a single lightning event resembles the distribution in the histogram of Figure 1.6, with between 1 and 5 being the most likely. The interval between these dart leaders is distributed between 10 ms to 240 ms with the majority occurring in 40 ms intervals as the histogram shows in Figure 1.6[10]. The length of these lightning channels is typically about 5 km from cloud to

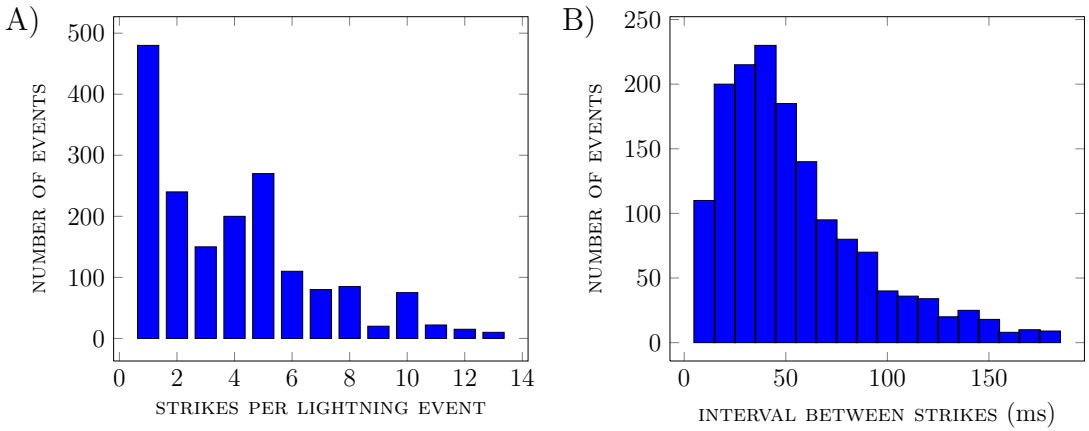

Figure 1.6: A) Histogram of dart leader frequency. B) Distribution of time between dart leader strikes.

ground[9] and the channel itself is only about one centimeter in diameter[5].

1.6.2 Characteristics of Thunder

As stated in the previous section, the electrical discharge of lightning heats the air in the channel to around 24,000 K. This heat causes the air in the channel to expand

rapidly. Almost 99% of energy from the lightning bolt is transferred into heat, leaving only 1% of energy to be turned into acoustic sound waves. The rapid expansion creates a channel pressure of around 10 to 100 atmospheres. This pressure results in the creation of a shock wave.[1] At 30,000 K the initial shock wave would begin traveling at approximately 3×10^3 m/s[5]. The shock wave itself moves through 3 stages: strong shock, weak shock, and acoustic. In early stages, the shock wave exhibits highly nonlinear behavior, which ultimately decays into an acoustic wave. The transition between the strong and weak shock stages is fairly distinct while the transition between the weak shock and acoustic phase is much more arbitrary[5]. The strong shock region is on the order of centimeters, while the weak shock stage stretches over several meters[9]. Weak shocks are also referred to as N-waves because they take the shape of the letter "N" as they travel (see Figure 1.1).

As these waves propagate through the atmosphere, they encounter other effects that influence their structure. Effects such as, but not limited to, geometrical spreading, interaction due to tortuousity, reflection, refraction, wind shear, temperature changes, absorption and dispersion. After a certain distance of travel called the "relaxation radius" by Few, the nonlinear N-wave evolves into a linear acoustic wave. This evolution can be seen in Figure 1.7. The acoustic phase decays over kilometers and beyond about 25 km from the channel an observer would likely not hear any thunder due in large part to refraction.

Only sound frequencies below 500 Hz typically reach an observer[3] and the most powerful thunder spectrum frequencies peak around 100 Hz according to Few[17] and Holmes[18]. Rakov[19] shows the peak spectrum power anywhere from subsonic (< 20 Hz) to 120 Hz and varying in time from initial higher frequencies that move to subsonic frequencies. The initial "crackling" heard in a lightning event is the multiple paths of the stepped leader traversing from the cloud to the ground before a streamer connects the channel to a certain part of the stepped leader and the strike occurs. To be clear, this work will only focus on the waves emitted from the initial return stroke and subsequent return strokes due to dart leaders.

1.6.3 Linear and Nonlinear Acoustics

In many applications of wave propagation, waves are modeled fundamentally by the basic wave equation. The basic wave equation is a hyperbolic partial differential equation that can be written as:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \tag{1.1}$$

[1] It is of note that this work will not be concerned with modeling the electrical process of lightning or thermodynamics of the initial expansion. The modeling will begin at an instantaneous pressure pulse, which is in itself an approximation of the result of the thermodynamic processes.

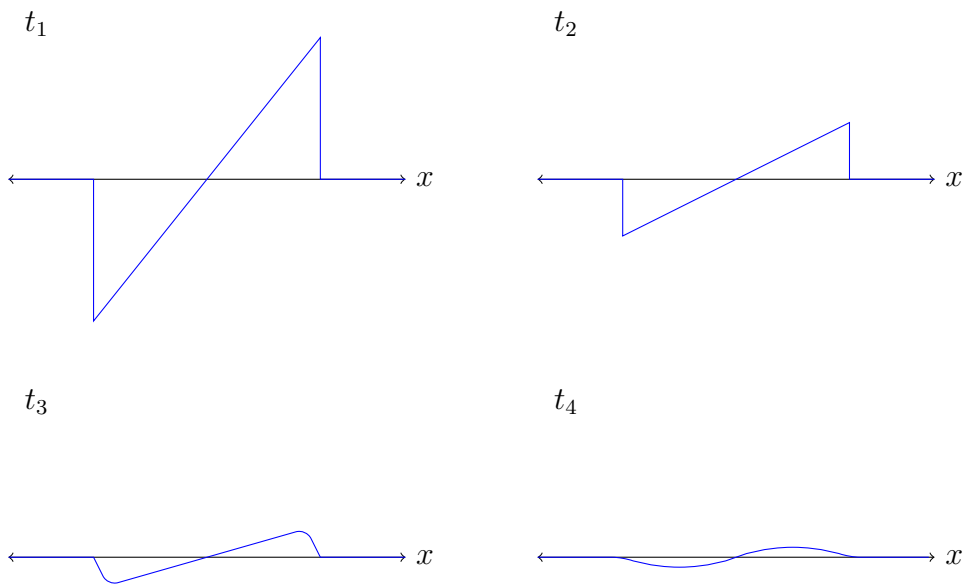

Figure 1.7: Times t_1 to t_4 depict the subsequent stages of an N-wave traveling to the right and decaying into an acoustic wave (scale is arbitrary as nonlinear waves will stretch over time).

where u is the position, t is time, and c is the speed of sound in the medium. When the wave amplitude is sufficiently small, temperature changes in the medium can be ignored. When temperature changes in the medium are ignored, c is considered a constant. With c a constant, this equation can be solved analytically, and modeling wave propagation is trivial. Equation 1.1 is homogeneous because it lacks a forcing term and does not include losses the wave may incur as it travels through the medium, such as absorption of the wave or geometrical spreading of the wave. These losses contribute to the decay of the wave as it propagates and they can be added into the equation.

Standard research in acoustics is typically focused on modeling linear waves which are low-amplitude disturbances where the linear wave equation is usually adequate. However, when the wave amplitude grows high enough such as it does initially in the event of thunder, the effects of nonlinearity become apparent, and linear assumptions fail, which means nonlinear terms excluded from the linear wave equation cannot be ignored. For example, wave steepening and wave deformations occur, such as shock waves, absorption increases, and the possibility cavitation may occur or solitons may arise. Nonlinear waves also interact with each other differently and are not generally superpositions as linear wave interactions may be described[20].

One paramount effect in nonlinear acoustics is the steepening of a waveform. This occurs due to the reality that high amplitude sections of a waveform travel faster than

low amplitude sections. The velocity of a wave can be described as[21]:

$$\left.\frac{dx}{dt}\right|_u = \beta u \pm c_0, \tag{1.2}$$

where the + sign is for an incoming wave and − sign for an outgoing wave, $\beta = 1+B/2A$ is the coefficient of nonlinearity[2], u is the particle velocity and c_0 is the small amplitude speed of sound in the medium. A visual representation of this phenomena can be seen in Figure 1.8. If a wave's amplitude is very small, this behavior can be

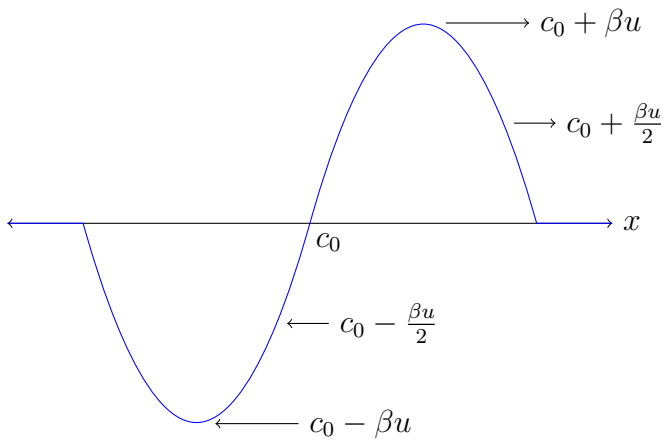

Figure 1.8: The effect of wave steepening due to the term βu where higher pressure sections of a wave travel faster than lower amplitude sections of the wave.

safely ignored. However, if a model is to account for waves with amplitudes that are sufficiently large, this nonlinear behavior cannot be disregarded. Wave steepening has an effect on the frequency of the traveling wave in that it tends to increase it.

The other paramount effect in nonlinear acoustics is *shock coalescence*. After a wave has steepened, weak shock theory states that the speed of the shock can be approximated by[23]:

$$U_{sh} = c_0 + \beta\frac{p_a + p_b}{2\rho_0 c_0}, \tag{1.3}$$

where p_a is the pressure ahead of the shock and p_b is the pressure behind the shock. This means that shocks with greater overall pressures $(p_a + p_b)$ will outrun weaker shocks and assimilate them. This is the process of shock coalescence. The effect that shock coalescence has on the traveling waveform is that it effectively deletes zero-crossing and lowers the fundamental frequency of the waveform. A depiction of this can be seen in Figure 1.9. Shock coalescence is one reason why frequencies in thunder higher than 500 Hz have difficulty making their way to an observer.

[2]The reader is referred to Reference [22] for a detailed description of the parameter B/A which is important in general nonlinear acoustics.

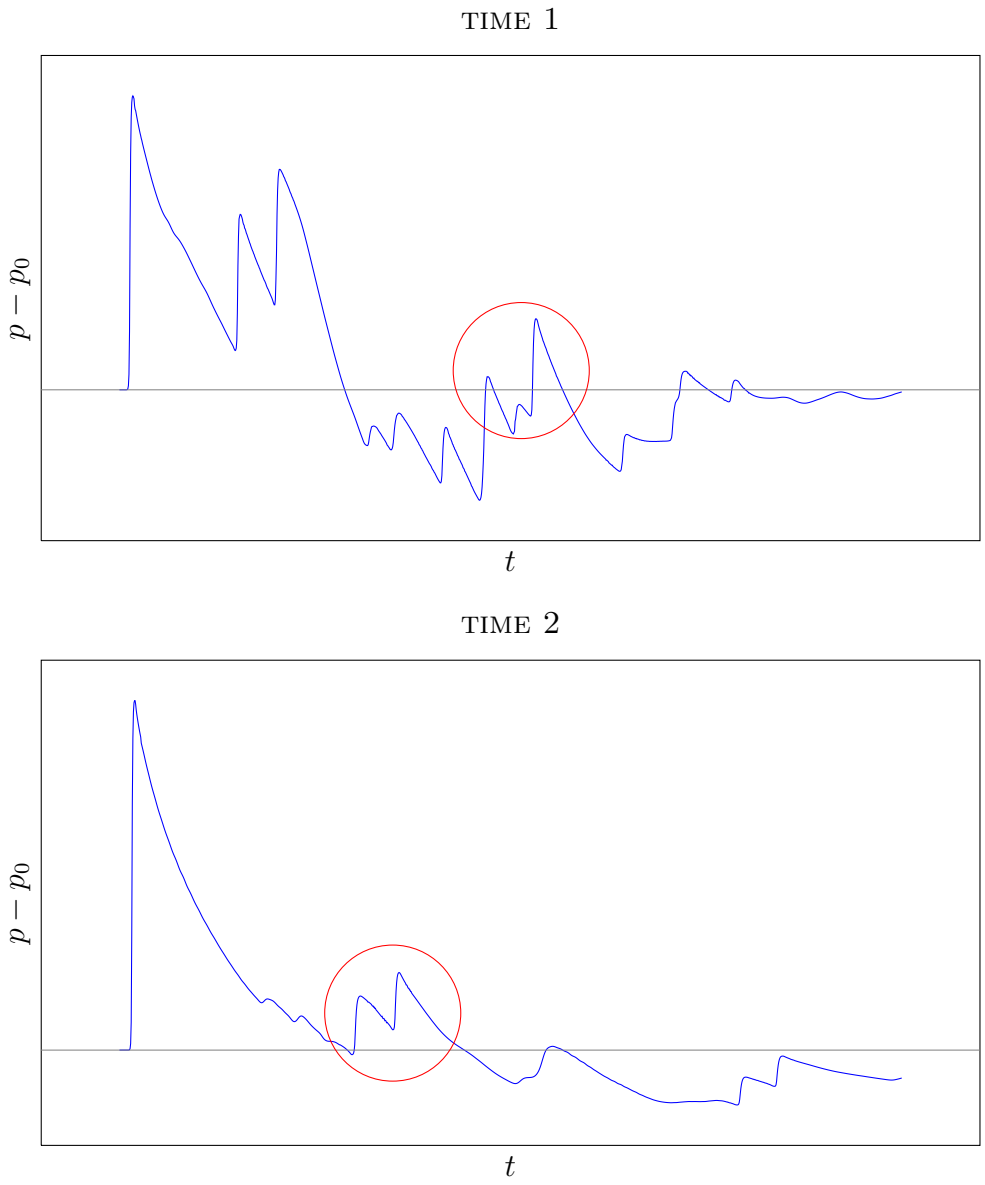

Figure 1.9: The effect of shock coalescence. One waveform is shown at TIME 1 and a later time, TIME 2. In this waveform it can be seen that the shocks in the red circle in TIME 1 have coalesced by TIME 2 and some zero-crossings have disappeared. It is apparent that shock coalescence has occurred in other locations of the waveform as well and that the energy has shifted toward the front.

1.6.4 Wave Attenuation

As waves propagate through air, the effects of absorption and dispersion contribute to the decay of the wave. In the three-dimensional world, acoustic waves in air spread spherically and losses will occur due to that geometrical expansion.

Thermoviscous absorption contributes two processes which are, the loss of molecular momentum due to collisions between molecules and the conduction of heat between molecules, known as shear viscous loss and thermal conduction loss, respectively. Thermoviscous absorption is known as *classical absorption*. There is an additional process that contributes to absorption in air known as bulk viscous loss, which when added to classical absorption is then known as *modified classical absorption*. Bulk viscosity is the loss due to the transfer of momentum into rotational and translational modes of molecular motion[15].

Another process which contributes to dispersion in air deals with energy losses due to the relaxation of different vibrating molecules. A certain time is required for a medium to establish equilibrium when a change in state occurs. This process is known as *molecular relaxation* and it depends highly upon the vibration of particular molecules. Air is comprised of 21% oxygen and 78% nitrogen which account for 99% of the molecules in air[15]. Therefore, molecular relaxation effects are largely associated with these two molecules and the energy dissipation depends particularly on the *relaxation frequency* of these molecules. The effects of molecular relaxation are discussed further in Chapter 2 where a justification toward rewriting the chosen model will take place.

1.6.5 Domain Size

Rather than take the approach of using WM-waves like most previous computer models for thunder, a more rigorous, CFD-type model solved by finite difference methods will be employed. This will allow the model to account for several effects of air in the atmosphere, as well as wave interactions. The disadvantage of this model will be the amount of computation necessary to scale the simulation to a realistically-sized domain.

A standard practice for modeling acoustic waves is to discretize the domain using between 10–20 points per the shortest wavelength (highest frequency) that should be accounted for. Therefore, the higher the frequencies of concern are, the more elements (or grid cells) will need to be accounted for in a particular size domain. After discretizing the domain into the amount of cells necessary, the pressure value can then be stored in each cell, as well as any other variables that are pertinent to the model.

Ordinarily, humans can hear frequencies between 20–20,000 Hz, which in wavelengths is 17.2–0.0172 meters. Using 10 points per wavelength (PPW), cell sizes as small as 0.00172 would be needed for the entire audible spectrum. If only frequencies

below 500 Hz are of concern, the wavelength is a much less demanding 0.688 meters, and below 200 Hz, 1.72 meters, corresponding to cell sizes of 0.0688 meters and 0.172 meters, respectively. For the size of the domain, average lengths of lightning channels are assumed to be 5 km and that thunder is generally inaudible over 25 km away from a strike. This gives a maximum three-dimensional domain of 5 km $\times$ 50 km $\times$ 50 km which would, at the time of this writing, be impossible to manage in any computer. A two-dimensional domain would still hold merit, as lightning channels do not vary as much horizontally as they do vertically, and since there is little interaction between waves on opposite sides of the strike in two dimensions, only one side of a strike would need to be modeled. Now when placing the observer closer to the strike, 8 km away for example, then the domain becomes a more manageable 5 km $\times$ 8 km. Yet, this domain size would still be substantial enough that modeling with normal computers would not be trivial. If deciding to only handle frequencies below 200 Hz using 10 PPW, with a short lightning channel of 2 km and an observer 2 km away, this gives roughly $11628^2 = 135,210,384$ cells necessary for computation. With 8 bytes necessary to store each cell, a grid that stores values for just one variable would take about 1GB of memory. Once other variables are considered in the model, the storage requirements become unwieldy for an average desktop computer fairly quickly. Therefore, it is apparent that the techniques discussed in Section 1.3 will be necessary to deal with this issue. A domain of the size $\sim$2–4 km $\times$ $\sim$2–4 km will be targeted in this work with frequencies below $\sim$500 Hz of concern.

1.7 Overview

This dissertation is comprised of six chapters and two appendices. This chapter briefly described the objectives of this work and previous work in the area of simulating thunder, as well as some other models that are used in nonlinear acoustics applications, leading into the model that was chosen for this work. In addition to exploring the physical characteristics of lightning and thunder, it discussed the topic of nonlinear acoustics to give a sense of how they might be modeled and what requirements a proper model should have and what it should account for.

Chapter 2 explains the equations of the chosen model in detail. The full model is described, as well as a model in which the effects of molecular relaxation due to oxygen are truncated. This is necessary in order to ease the restrictions on the time step length of the model during runtime for some larger simulations that are presented in this work and it is shown that only the effects of molecular relaxation due to nitrogen are significant to thunder propagation. A slight rewrite of the equations is also emphasized for correct application of the model equations.

Chapter 3 details the Runge-Kutta scheme used on the time derivative and the WENO, DRP and hybrid methods used on the spatial derivatives which are used to numerically solve the equations of the model. Their stability is also examined.

Chapter 4 discusses details concerning the implementation of the model in the computer. It discusses the geometry of lightning channels and how they are inserted as sources into the model in an efficient manner. Pseudocode is given for a basic implementation of the model that is structured in such a way that it can be extended into an AMR framework. It also discusses how the model is extended by AMR in order to counteract its computational magnitude while attempting to retain its possible level of accuracy.

Chapter 5 analyzes the hybrid scheme, newly added effects, the results obtained by the AMR implementation of the model, the choice of methods, and the performance benefits of the modifications to the selected model. A phenomenological study is also conducted on example lightning events.

Chapter 6 summarizes the conclusions of the work. It also suggests possible improvements and extensions to the work.

Appendix A lists MATLAB code for generating random lightning channel geometries as well as code for drawing the lightning channel on the grid using the generated list of segments as described in Chapter 4. Appendix B documents source code which can be used to promptly implement the basic model in MATLAB as depicted in this work using a static mesh.

Chapter 2

Model

2.1 Introduction

This chapter is based on the work of Wochner[8] and the reader is referred to his work for further information, including the derivation and verification of the model. Since the model is an integral part of this work and modifications will be made in the implementation, it will be discussed in detail. Wochner built upon a previous model by Pierce[15] by adding in the effects of molecular relaxation for oxygen and nitrogen. Wochner describes molecular relaxation as a dominant process of absorption in audible frequencies. Therefore, the model will be detailed with both of these effects included for completeness, but molecular relaxation due to oxygen will be justifiably excluded as its effects are not as significant to thunder propagation. Also, the effect of oxygen relaxation has more demanding restrictions on the size of time steps when numerically solving the model. After implementing the model with molecular relaxation, it is trivial to exclude the effects of one molecule or both as will be detailed in Section 2.2.2.

2.2 Wochner's Model

The Navier-Stokes equations used in fluid dynamics are the basis for the model. The Navier-Stokes equations are generally known as extensions to the Euler equations in which viscosity has been added. These equations are capable of accounting for the nonlinear waves as well as linear waves that occur after lightning events in air. Modified classical absorption is included, as well as a modification to the original model to account for refraction of waves in the atmosphere. This model exhibits cylindrical spreading due to its confinement to two dimensions.[1] However, it essentially exists on

[1]When using a tortuous lightning channel source, a loss of axisymmetry is incurred and therefore, the addition of spherical spreading through the inclusion of a geometrical source term would be unnatural. Consequently, geometrical spreading in only two dimensions is tolerated. Li[24] has

a plane in three-dimensions due to the units used being the three-dimensional units, i.e. Pascals.

2.2.1 Full Model

For a two-dimensional domain in a Cartesian coordinate system, Wochner's model is a set of coupled equations which can be written succinctly in the form:

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = \mathbf{H}, \tag{2.1}$$

where the vectors for $\mathbf{w}$, $\mathbf{F}$, $\mathbf{G}$, and $\mathbf{H}$ are

$$\mathbf{w} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho s_{fr} \\ \rho T_{N_2} \\ \rho T_{O_2} \end{pmatrix}, \ \mathbf{F} = \begin{pmatrix} \rho u \\ \rho u^2 \\ \rho uv \\ \rho u s_{fr} \\ \rho u T_{N_2} \\ \rho u T_{O_2} \end{pmatrix}, \ \mathbf{G} = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 \\ \rho v s_{fr} \\ \rho v T_{N_2} \\ \rho v T_{O_2} \end{pmatrix}, \tag{2.2}$$

and

$$\mathbf{H} = \begin{pmatrix} 0 \\ -\frac{\partial p}{\partial x} + \mu_B \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial x \partial y} \right) + \mu \left(\frac{\partial \phi_{xx}}{\partial x} + \frac{\partial \phi_{xy}}{\partial y} \right) \\ -\frac{\partial p}{\partial y} + \mu_B \left(\frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 u}{\partial y \partial x} \right) + \mu \left(\frac{\partial \phi_{yx}}{\partial x} + \frac{\partial \phi_{yy}}{\partial y} \right) \\ \sigma_s - \sum_\nu \frac{\rho}{T_\nu} c_{v\nu} \frac{DT_\nu}{Dt} + \nabla \cdot \left(\frac{\kappa}{T} \nabla T \right) \\ \frac{\rho}{\tau_{N_2}} (T - T_{N_2}) \\ \frac{\rho}{\tau_{O_2}} (T - T_{O_2}) \end{pmatrix}. \tag{2.3}$$

In Equations 2.1–2.3, x and y are the dimensions of the domain, t is time, ρ is the density, u and v each component of the velocity vector, s_{fr} the frozen entropy, T_ν the apparent vibration temperature for the ν-type molecule where the ν-type molecules are $\nu = O_2$ (for oxygen), and $\nu = N_2$ (for nitrogen), p the pressure, T the temperature, $\mu = 1.846 \times 10^{-5}$ kg/(m·s) and $\mu_B = 0.6 \cdot \mu$ the shear viscosity and bulk viscosity, respectively, ϕ the rate-of-shear tensor, κ the coefficient of thermal conduction given as 2.624×10^{-2} W/(m·K), $c_{v\nu}$ the specific heat at constant volume associated with the ν-type molecule, σ_s a variable representing source terms, and lastly, τ_ν the relaxation time of the ν-type molecule.

The source term σ_s is defined as

$$\sigma_s = \frac{\mu_B}{T} (\nabla \cdot \mathbf{v})^2 + \frac{\mu}{2T} \sum_{ij} \phi_{ij}^2 + \frac{\kappa}{T^2} (\nabla T)^2 + \frac{\rho}{T} \sum_\nu A_\nu \frac{DT_\nu}{Dt}. \tag{2.4}$$

information on obtaining geometrical source terms for the Euler equations.

In Equation 2.4, the rate-of-shear tensor is defined as:

$$\phi_{ij} = \frac{\partial z_i}{\partial q_j} + \frac{\partial z_j}{\partial q_i} - \frac{2}{3}\nabla \cdot \mathbf{z}\delta_{ij}, \tag{2.5}$$

where $q_x = x$ and $q_y = y$, z_i is the i^{th} component of the velocity and δ_{ij} is the Kronecker delta, where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ if $i \neq j$. In Equation 2.5, the subscripts $i = x, y$ and $j = x, y$ should not be confused with the typical notation where subscripts using x and/or y denote partial derivatives. For example, in the case of ϕ_{ij}, ϕ_{xx} would denote the rate-of-shear associated with the "xx-direction", and not the partial second derivative of ϕ with respect to x. To be more clear, the four cases for ϕ_{ij} simplify to:

$$\begin{aligned} \phi_{xx} &= \frac{4}{3}\left(\frac{\partial u}{\partial x}\right) - \frac{2}{3}\left(\frac{\partial v}{\partial y}\right), \\ \phi_{yx} = \phi_{xy} &= \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}, \\ \phi_{yy} &= \frac{4}{3}\left(\frac{\partial v}{\partial y}\right) - \frac{2}{3}\left(\frac{\partial u}{\partial x}\right). \end{aligned}$$

Also in Equation 2.4:

$$A_\nu = \left(\frac{T}{T_\nu} - 1\right) c_{v\nu}, \tag{2.6}$$

where

$$c_{v\nu} = \frac{n_\nu}{n} R \left(\frac{T_\nu^*}{T_\nu}\right)^2 e^{-\frac{T_\nu^*}{T_\nu}}. \tag{2.7}$$

In Equation 2.6, A_ν is the affinity associated with the ν-type molecule, n_ν/n the fraction of all molecules in air of type ν, given as $n_{T_{O_2}}/n = 0.21$ and $n_{T_{N_2}}/n = 0.78$[15], $R = 287.06$ J/(kg·K) the gas constant, and T_ν^* the molecular constant of the molecule ν, given as $T_{O_2}^* = 2239$ and $T_{N_2}^* = 3352$[15]. In Equation 2.3 and 2.4 the relaxation equation is defined as

$$\frac{DT_\nu}{Dt} = \frac{1}{\tau_\nu}(T - T_\nu).$$

To close the set of coupled equations, equations for the pressure and temperature must be defined which relate them to the density. These are called the *equations of state*. The equations for pressure and temperature used are:

$$p = c^2\left[(\rho - \rho_0) + \left(\frac{\gamma - 1}{2\rho_0}\right)(\rho - \rho_0)^2 + \left(\frac{\rho\beta T}{c_p}\right)_0 (s_{fr} - s_{fr_0})\right] + p_0, \tag{2.8}$$

and

$$T = T_0 e^{(s_{fr} - s_{fr_0} - R\ln[\rho_0/\rho])/c_\nu}, \tag{2.9}$$

where c is the speed of sound in air and will be a function of height denoted as $c = c_0 + (-6 \times 10^{-3})y$ m/s where $c_0 = 343$ m/s. Also, in Equation 2.8[2], ρ_0 is the ambient density given as 1.21 kg/m^2, $\gamma = 1.402$, $\beta T = 1$ (note β is not the coefficient of nonlinearity in this case, but the coefficient of thermal expansion and the Ideal Gas Law implies $\beta T = 1$[7]), s_{fr_0} the ambient entropy given as 0, p_0 the ambient pressure given as 101325 Pa, $T_0 = 293.16$ K the ambient temperature, $c_v = 720.4$ J·kg^{-1}·K^{-1} the specific heat of air at constant volume, and $c_p = 1010$ J·kg^{-1}·K^{-1} the specific heat of air at constant pressure. Equation 2.8 is chosen from Pierce[15] instead of the equation for pressure recommended by Wochner. It has been chosen in order to exploit the c term to include wave refraction in the model, as the speed of sound varies with altitude.

The formula for the relaxation times of ν-type molecules is:

$$\tau_\nu = \frac{1}{2\pi f_\nu},$$

where the relaxation frequencies are given as[3]

$$f_{N_2} = \frac{p_0}{p_{ref}} \left[\left(\frac{T_{ref}}{T_0}^{1/2} \right) (9 + 280he^{-\eta}) \right]. \tag{2.10}$$

and

$$f_{O_2} = \frac{p_0}{p_{ref}} \left(24 + 4.04 \times 10^4 h \frac{0.02 + h}{0.391 + h} \right), \tag{2.11}$$

In Equations 2.10 and 2.11:

$$h = \frac{(RH)p_{vp}}{p_0}, \tag{2.12}$$

$$\eta = 4.17 \left[\left(\frac{T_{ref}}{T_0}^{1/3} \right) - 1 \right], \tag{2.13}$$

and

$$p_{vp} = p_{ref} 10^{\varphi}, \tag{2.14}$$

where

$$\begin{aligned} \varphi = {} & 10.79586 \left(1 - \left(\frac{273.16}{T_0} \right) \right) - 5.02808 \log_{10} \frac{T_0}{273.16} \\ & + 1.50474 \times 10^{-4} \left(1 - 10^{-8.29692 \left(\frac{T_0}{273.16} - 1 \right)} \right) \\ & - 4.2873 \times 10^{-4} \left(1 - 10^{-4.76955 \left(\frac{273.16}{T_0} - 1 \right)} \right) - 2.2195983. \end{aligned}$$

[2] In Equation 2.8 $()_0$ means that the partial derivatives the quantity was derived from are evaluated at constant entropy.

[3] The relaxation frequencies are reversed in Wochner[8] as is apparent in the citation in which they were obtained: Bass *et al.*[25].

In Equation 2.12, h is the fraction of molecules in the air that is H_2O, RH is the percent of relative humidity (set to be a number between 0 and 100, which can be varied but will be taken as 20), and in Equation 2.13 and 2.14, $T_{ref} = 293.16$ K and $p_{ref} = 101325$ Pa. Once τ_{N_2} and τ_{O_2} are calculated, they will be considered constants.

When initializing the variables of the model in the solution vector $\mathbf{w}$, the ambient values given should be used. However, initial values will also need to be given to the last two elements in $\mathbf{w}$ involving T_ν which are given as $T_{\nu_0} = T_0$. A summarized list of values used for the constants can be located in Table 2.1.

Constant	Value	Units
κ	2.624×10^{-2}	W/(m·K)
c_0	343	m/s
c	$c_0 + (-6 \times 10^{-3})y$	m/s
γ	1.402	-
T_{ref}	293.16	K
p_{ref}	101325	Pa
c_v	720.4	$J \cdot kg^{-1} \cdot K^{-1}$
c_p	1010	$J \cdot kg^{-1} \cdot K^{-1}$
μ	1.846×10^{-5}	kg/(m·s)
μ_B	$0.6 \cdot \mu$	kg/(m·s)
$n_{T_{O_2}}/n$	0.21	-
$n_{T_{N_2}}/n$	0.78	-
RH	20	-
p_0	101325	Pa
T_0	293.16	K
ρ_0	1.21	kg/m^2
R	287.06	J/(kg·K)
$T^*_{N_2}$	3352	K
$T^*_{O_2}$	2239	K
s_{fr_0}	0	-
$T_{\nu 0}$	T_0	K
β	$1/T$	K

Table 2.1: Table of constants and their values for use in the model.

2.2.2 Truncated Model

Before naively implementing Wochner's full model at the scale in which this work is intended, it is worthwhile to investigate the effects of molecular relaxation further, as their inclusion significantly increases the amount of computation required. As

mentioned in Chapter 1 the effects of molecular relaxation depend preponderantly on the specific relaxation frequencies of each molecule. Each molecule has an associated attenuation coefficient that varies according to wave frequency and these coefficients are considered to act independently of one another. Figure 2.1 shows a plot of these attenuation coefficients along with the coefficient of modified classical absorption using the constants of the model described in Section 2.2.1. It can be seen that the attenuation (in Nepers/m) due to relaxation of nitrogen reaches its highest impact around 138 Hz, while for oxygen, it occurs around 10556 Hz, and modified classical absorption has its highest effect on very high frequencies (10^5 Hz). The linear composition of these attenuation effects is also plotted to show their combined effect. These effects are small, but they become significant in long distance propagation of waves. Since frequencies above 500 Hz have little to do with the propagation of thunder, the effect molecular relaxation due to oxygen should have on the problem is minor and therefore comfortably excluded from the model. This also reduces the amount of storage and calculation necessary for running the model. Figure 2.2 shows the effect the relative humidity parameter has on the relaxation frequencies of nitrogen and oxygen.

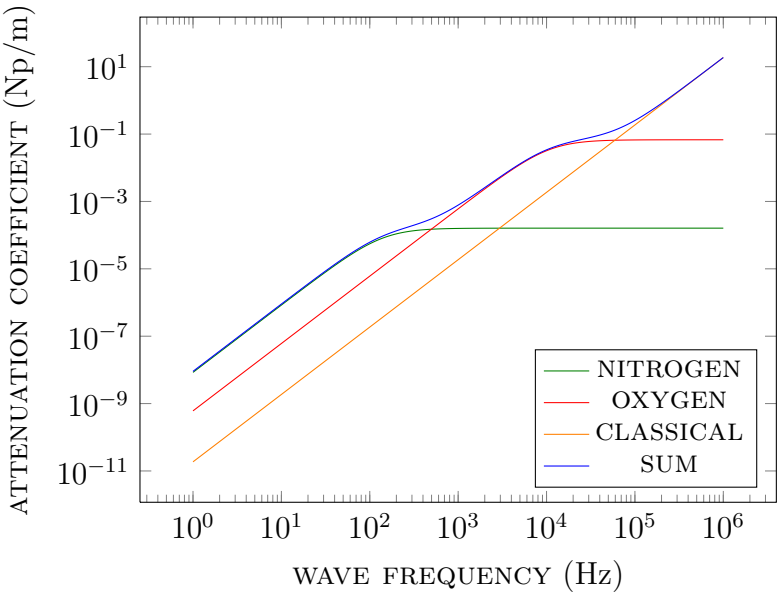

Figure 2.1: Plot displaying attenuation coefficients according to wave frequency along with their linear combination. The coefficient for nitrogen is in green, oxygen in red, modified classical absorption in orange and their linear combination in blue. It is apparent that the effect for nitrogen dominates frequencies below 500 Hz.

To exclude molecular relaxation due to oxygen, two actions are taken. The first is

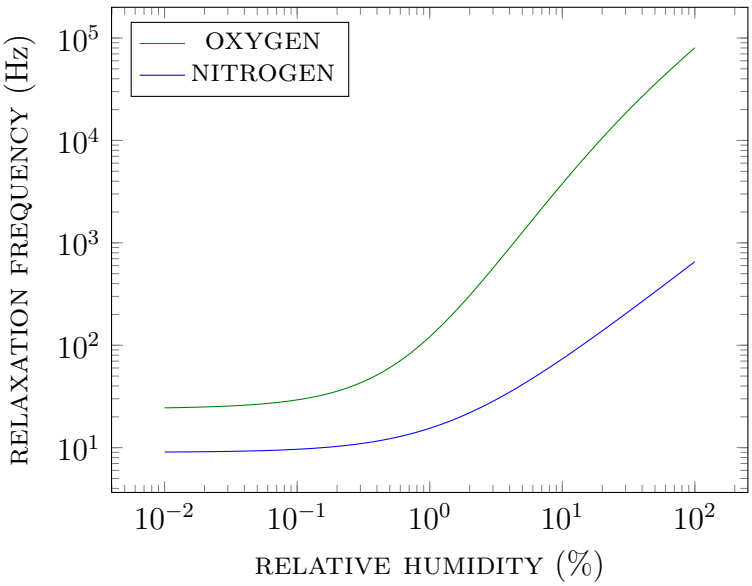

Figure 2.2: Plot of the effect relative humidity has on the relaxation frequencies of nitrogen and oxygen. The relaxation frequency for nitrogen is in blue and oxygen in green.

to set $c_{O_2 v} = 0$ in Equation 2.7, which removes a term from the summation in Equation 2.3. From Equation 2.6, another consequence is that $A_{O_2} = 0$ and a summation term is removed from Equation 2.4. The second action is to then omit the element of each vector in $\mathbf{w}$, $\mathbf{F}$, $\mathbf{G}$, and $\mathbf{H}$ that involves T_{O_2}.

At this point, a slight rewriting of the model equations will also occur to better suit the numerical methods which will be used to solve. This rewriting of the equations will help remedy oscillations that the reader may experience if the model was implemented exactly how it was previously written. To do so, one of the constitutive equations that was used to derive the model is revisited. This equation is the Navier-Stokes equation (sometimes called the momentum equation), which is written as[26][4]

$$\rho \frac{D\mathbf{v}}{Dt} = -\nabla p + \nabla(\mu_B \nabla \cdot \mathbf{v}) + \mu \sum_{ij} \mathbf{e}_i \frac{\partial \phi_{ij}}{\partial x_j}. \tag{2.15}$$

In two dimensions, Equation 2.15 has x and y components. Therefore, when the constitutive equations are recast in conservative form, Equation 2.15 is rewritten as:

$$\begin{aligned} &\frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2)}{\partial x} + \frac{\partial(\rho uv)}{\partial y} \\ &\qquad = -\frac{\partial p}{\partial x} + \mu_B \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial x \partial y} \right) + \mu \left(\frac{\partial \phi_{xx}}{\partial x} + \frac{\partial \phi_{xy}}{\partial y} \right), \end{aligned} \tag{2.16}$$

[4]Note that if an external force such as wind is desired, this force can be added into the momentum equation as an extra term, f_w, for example, as will be done in Chapter 5.

and

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho vu)}{\partial x} + \frac{\partial(\rho v^2)}{\partial y} = -\frac{\partial p}{\partial y} + \mu_B \left(\frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 u}{\partial y \partial x} \right) + \mu \left(\frac{\partial \phi_{yx}}{\partial x} + \frac{\partial \phi_{yy}}{\partial y} \right). \tag{2.17}$$

In Equation 2.15, the $-\nabla p$ term will be moved to the left-hand side and grouped over their respective derivatives. Equations 2.16 and 2.17 then become:

$$\frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2 + p)}{\partial x} + \frac{\partial(\rho uv)}{\partial y} = \mu_B \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial x \partial y} \right) + \mu \left(\frac{\partial \phi_{xx}}{\partial x} + \frac{\partial \phi_{xy}}{\partial y} \right), \tag{2.18}$$

and

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho vu)}{\partial x} + \frac{\partial(\rho v^2 + p)}{\partial y} = \mu_B \left(\frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 u}{\partial y \partial x} \right) + \mu \left(\frac{\partial \phi_{yx}}{\partial x} + \frac{\partial \phi_{yy}}{\partial y} \right), \tag{2.19}$$

respectively. In this form, the $-\nabla p$ term will be calculated using the high order methods for spatial derivatives introduced in Chapter 3, some of which are able to stably propagate discontinuities and avoid oscillations. In the previous form (Equations 2.16–2.17), it is possible that oscillations would arise at implementation regardless of the method for calculating spatial derivatives. If the full model is desired, this rewrite should be done as well.

The truncated model is again stated as:

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = \mathbf{H}, \tag{2.20}$$

where the vectors for $\mathbf{w}$, $\mathbf{F}$, $\mathbf{G}$, and $\mathbf{H}$ are

$$\mathbf{w} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho s_{fr} \\ \rho T_{N_2} \end{pmatrix}, \ \mathbf{F} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho u s_{fr} \\ \rho u T_{N_2} \end{pmatrix}, \ \mathbf{G} = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho v s_{fr} \\ \rho v T_{N_2} \end{pmatrix}, \tag{2.21}$$

and

$$\mathbf{H} = \begin{pmatrix} 0 \\ \mu_B \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial x \partial y} \right) + \mu \left(\frac{\partial \phi_{xx}}{\partial x} + \frac{\partial \phi_{xy}}{\partial y} \right) \\ \mu_B \left(\frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 u}{\partial y \partial x} \right) + \mu \left(\frac{\partial \phi_{yx}}{\partial x} + \frac{\partial \phi_{yy}}{\partial y} \right) \\ \sigma_q - \frac{\rho}{T_{N_2}} c_{vN_2} \frac{DT_{N_2}}{Dt} + \nabla \cdot \left(\frac{\kappa}{T} \nabla T \right) \\ \frac{\rho}{\tau_{N_2}} (T - T_{N_2}) \end{pmatrix}. \tag{2.22}$$

Here,

$$\sigma_q = \frac{\mu_B}{T}(\nabla \cdot \mathbf{v})^2 + \frac{\mu}{2T}\sum_{ij}\phi_{ij}^2 + \frac{\kappa}{T^2}(\nabla T)^2 + \frac{\rho}{T}A_{N_2}\frac{DT_{N_2}}{Dt}. \tag{2.23}$$

Values for the variables and constants in this model will use the same as noted in the full model and Equations 2.8 and 2.9 will again be used for pressure and temperature, respectively.

2.3 Summary

In this section the model selected for this work was described. It is a two-dimensional model in Cartesian coordinates based on the Navier-Stokes equations. It includes modified classical absorption effects as well as molecular relaxation due to nitrogen and oxygen in the full model. A truncated model excluding the effect of molecular relaxation due to oxygen was also included which is the model that will be implemented in this work. This model was also slightly rewritten to better exploit the methods that will be used for solving the model. In Equation 2.20 it can be seen that each vector is contained within a particular derivative. This allows for various explicit time-marching numerical methods to be used in solving the model's coupled set of equations. The method that will be used to solve is Runge-Kutta scheme in time and a hybrid of two different methods in space. These methods are described in Chapter 3.

Chapter 3

Methods

3.1 Introduction

The model described in Chapter 2 has no known analytical solutions. Therefore, it will be solved (or approximated rather) numerically by discretizing it in time and space. A crucial aspect of this work is efficiency in solving the model equations. Therefore, a finite difference scheme will be used as they are typically more efficient than finite element or finite volume methods.

The model is a set of coupled equations with two equations of state that close the set. Once the system is given initial conditions it will be "marched" through time using an explicit method where each equation must be solved at each time step. To handle the time derivative, a *total variation diminishing* (TVD) Runge-Kutta scheme is employed. Although there are other numerical methods for solving through time, such as the Godunov[27], Lax-Wendroff[28] or MacCormack[29] schemes, they are all second-order accurate and not as generalizable or as common as Runge-Kutta schemes.

For the spatial derivatives, care must be taken to solve by methods in which nonlinear, i.e. discontinuous, waves are minimally compromised. Shu[30] has demonstrated the advantage in using high-order methods for CFD for resolving features that low-order methods could not produce, with even less expensive computations than using low-order methods on higher resolution grids. Much of CFD is focused on the use of high-order finite volume methods, such as the *discontinuous Galerkin* scheme, or finite volume *weighted essentially non-oscillatory* (WENO) scheme. For high-order finite differences, the finite difference WENO scheme is a popular choice and its accuracy and performance are adjustable. With these qualities and the finite difference WENO scheme's capability of stably propagating discontinuous waves, it is therefore chosen for use to calculate the spatial derivatives in the model.

The WENO scheme will be the most computationally demanding aspect in the implementation of the model. For that reason, a centered finite difference method

called the *dispersion-relation preserving* (DRP) scheme, is employed in coalition with the WENO scheme, resulting in a hybrid scheme of the two. The DRP scheme involves much less computation but fails to propagate shocks well. The WENO scheme will be assured use in areas containing discontinuities by calculating a smoothness indicator over the grid to determine where the highest perturbations of values on the grid are located. Third-order accurate and fifth-order accurate WENO schemes are described for service in the hybrid scheme. For a more general discussion on other high-order schemes for use in CFD, the reader is referred to Shu and Juan[31].

3.2 Time Derivative

The first step to numerically solving Equation 2.1 is to isolate the time derivative on the left-hand side. Doing so results in the equation rewritten in the form:

$$\frac{\partial \mathbf{w}}{\partial t} = -\frac{\partial \mathbf{F}}{\partial x} - \frac{\partial \mathbf{G}}{\partial y} + \mathbf{H}. \tag{3.1}$$

The vector $\mathbf{w}$ will then be known as the *solution vector*. It can be seen from Equation 2.2 that the vectors $\mathbf{F}$ and $\mathbf{G}$ can be derived from the primitive variables in $\mathbf{w}$. The general procedure is to step forward in time, update the solution vector by computing the right-hand side of Equation 3.1, update the remaining vectors and the equations of state, and then repeat the steps until the desired length of simulation time is reached. Updating the system a full time step is done by performing every stage of the selected Runge-Kutta scheme.

3.2.1 Runge-Kutta Schemes

For an example initial value problem written as:

$$\frac{\partial u}{\partial t} = L(u), \tag{3.2}$$

where u is considered the solution in this case, and the example stands separate from the actual model, a Runge-Kutta scheme can be used to solve. In the case of the actual model, Equation 3.1 is analagous to Equation 3.2. In Gottlieb[32] it has been shown that if a non-TVD Runge-Kutta method for a hyperbolic problem (such as this selected model) were used, the results may be oscillatory if the L in Equation 3.2 is nonlinear. The L in the selected model is, in fact, nonlinear. Therefore, a TVD Runge-Kutta scheme is used. Although higher order accurate TVD Runge-Kutta schemes exist, only the second-order and third-order accurate versions are employed in this work. When using a Runge-Kutta scheme, storage can become an issue for large scale simulations. Therefore, low-storage Runge-Kutta methods attempt to limit the amount of storage to the equivalent of storing two copies of the solution

vector. Without any modifications, this is already the case of the third-order and second-order versions. The solution vector (u in Equation 3.2) at the current time step must be stored in the computer's memory as well as the solution vector for the current stage in the Runge-Kutta.

3.2.1.1 Third-Order

A third-order accurate TVD Runge-Kutta scheme is said to have three stages and is described as follows[32]:

$$u^{(1)} = u^n + \Delta t L(u^n) \tag{3.3}$$

$$u^{(2)} = \frac{3}{4}u^n + \frac{1}{4}u^{(1)} + \frac{1}{4}\Delta t L\left(u^{(1)}\right) \tag{3.4}$$

$$u^{n+1} = \frac{1}{3}u^n + \frac{2}{3}u^{(2)} + \frac{2}{3}\Delta t L\left(u^{(2)}\right). \tag{3.5}$$

3.2.1.2 Second-Order

A second-order accurate TVD Runge-Kutta scheme is said to have two stages and is described as follows[32]:

$$u^{(1)} = u^n + \Delta t L(u^n) \tag{3.6}$$

$$u^{n+1} = \frac{1}{2}u^n + \frac{1}{2}u^{(1)} + \frac{1}{2}\Delta t L\left(u^{(1)}\right). \tag{3.7}$$

3.3 Spatial Derivatives

At each stage of the Runge-Kutta scheme, the spatial derivatives on the right-hand side of Equation 3.1 must be recalculated. This is done by the use of a hybrid scheme where the subschemes in use will be the WENO scheme and DRP scheme. The WENO scheme has the ability to propagate shocks well, but it is more computational in comparison to the DRP scheme which does not propagate shocks well and exhibits oscillation at discontinuities which occur in shock waves as well as reflecting boundary conditions.

3.3.1 WENO Scheme

The WENO scheme introduced by Shu[33] will be described first. It has a more complex implementation than the DRP scheme. These schemes calculate the spatial derivatives by using a specific set of *stencils*. Stencils are the set of points on the grid surrounding the grid cell in which the derivative is to be calculated. In an n-point method for calculating a spatial derivate, the number of points surrounding the current grid cell to be used in the calculation is n. The stencil is considered

the entire set of the n points. In the WENO method, more than one set of stencils is used in the calculation. Each set of stencils will contain an arbitrary, but fixed, number of points around the grid cell. The advantage and entire basis of a WENO scheme is its ability to effectively leave stencils out of the calculation by calculating the smoothness of the data in each stencil, and then attaching a weight to them. If the stencil contains a discontinuity, it will be weighted very small and effectively left out of the calculation (avoiding oscillation). If the stencil contains smooth data, it will be weighted higher and maintain a part of the calculation. If smooth data exists in each stencil, an optimal weighting of the set of stencils will be used for the calculation. The WENO scheme has the burden of calculating the weights of the stencils, which regular n-point schemes do not have. Therefore, the WENO scheme has a higher computational magnitude than such schemes.

The third-order and fifth-order accurate WENO schemes are detailed here. The reader is referred to Shu[33][34] for a more general discussion on WENO schemes in which there exists schemes beyond fifth-order. In the WENO scheme, the parameter k denotes the number of stencils that make up the convex combination of stencils. For the third-order scheme, $k = 2$, and for the fifth-order scheme, $k = 3$. For higher order schemes, $k > 3$, the amount of computation necessary increases dramatically. To describe the WENO scheme, an example one-dimensional single differential equation:

$$\frac{\partial u}{\partial t} = -\frac{\partial f(u)}{\partial x}, \tag{3.8}$$

separate from the model will be used. For the rest of this section it is assumed that $\partial f(u)/\partial u \geq 0$, meaning information will be exclusively traveling in the positive direction. The computational domain is first discretized into a uniform mesh of X cells $x_i = i\Delta x$, $i = 1, 2, \ldots, X$, with Δx cell sizes. All initial conditions are assumed to have been established. An ordinary differential equation capable of approximating Equation 3.8 where $u_i(t)$ is a conservative approximation to $u(x_i, t)$ is given by:

$$\frac{du_i(t)}{dt} = -\frac{1}{\Delta x}(\hat{f}_{i+1/2} - \hat{f}_{i-1/2}). \tag{3.9}$$

The numerical flux $\hat{f}_{i+1/2}$ is defined as:

$$\hat{f}_{i+1/2} = \sum_{q=0}^{k-1} \bar{\omega}_q \bar{f}^{(q)}_{i+1/2}, \tag{3.10}$$

with

$$\bar{f}^{(q)}_{i+1/2} = \sum_{r=0}^{k-1} \bar{c}_{q,r} f_{i-q+r}, \tag{3.11}$$

where $\bar{c}_{q,r}$ are constants. These $\bar{c}_{q,r}$ are listed in Table 3.1 for $k = 2$ (third-order) and Table 3.2 for $k = 3$ (fifth-order). The nonlinear weights, (i.e. weights used when

q	$r=0$	$r=1$
0	1/2	1/2
1	−1/2	3/2

Table 3.1: Table of constants for $\bar{c}_{q,r}$, for $k=2$.

q	$r=0$	$r=1$	$r=2$
0	1/3	5/6	−1/6
1	−1/6	5/6	1/3
2	1/3	−7/6	11/6

Table 3.2: Table of constants for $\bar{c}_{q,r}$, for $k=3$.

the medium contains a discontinuity in the stencils) $\bar{\omega}_q$, are given as:

$$\bar{\omega}_q = \frac{\tilde{\omega}_q}{\sum_{l=0}^{k-1} \tilde{\omega}_l}, \tag{3.12}$$

and

$$\tilde{\omega}_s = \frac{\zeta_s}{(\epsilon + \beta_s)^2}, \quad \text{for } s = q, l. \tag{3.13}$$

Here ϵ will be set to 10^{-6} to avoid a possible zero in the denominator. It is required that:

$$\bar{\omega}_r \geq 0 \quad \text{and} \quad \sum_{r=0}^{k-1} \bar{\omega}_r = 1, \tag{3.14}$$

for stability and consistency. The linear weights, (i.e. weights used when the medium is continuous in the stencils) ζ_s, are given in Table 3.3. The smoothness indicators

k	$s=0$	$s=1$	$s=2$
2	2/3	1/3	
3	3/10	6/10	1/10

Table 3.3: Table of values for ζ_s for $k=2,3$.

β_s are defined as[33]:

$$\beta_0 = (f_{i+1} - f_i)^2, \tag{3.15}$$
$$\beta_1 = (f_i - f_{i-1})^2, \tag{3.16}$$

for $k = 2$. For $k = 3$ the β_s are:

$$\beta_0 = \frac{13}{12}(f_i - 2f_{i+1} + f_{i+2})^2 + \frac{1}{4}(3f_i - 4f_{i+1} + f_{i+2})^2, \tag{3.17}$$

$$\beta_1 = \frac{13}{12}(f_{i-1} - 2f_i + f_{i+1})^2 + \frac{1}{4}(f_{i-1} - f_{i+1})^2, \tag{3.18}$$

$$\beta_2 = \frac{13}{12}(f_{i-2} - 2f_{i-1} + f_i)^2 + \frac{1}{4}(f_{i-2} - 4f_{i-1} + 3f_i)^2. \tag{3.19}$$

Note that f_i is used as an abbreviation of $f(u_i(t))$ and the combination of stencils in the described WENO scheme in this section is biased to the left because it was assumed that information was moving in a strictly positive direction. This technique is known as *upwind biasing*.

For long-term propagation of shock waves the WENO scheme should be upwind biased, and will be described here as such. Upwind biasing is a modification imposed upon numerical schemes to place more points in the stencil used for the calculation behind a wave in the direction it is traveling. Although in the previous section the combination of stencils was biased to the left, it is desirable to bias the stencils to the right in the case of information traveling in a negative direction as well. This is done by using a technique known as *flux splitting*. There are several different methods of flux splitting (Roe[35], Lax-Friedrichs[33], Osher[36], Van Leer[37], Donat-Marquina[38], etc.) and they can have large effects if the numerical method used for solving is of low order, but in a high order scheme, different flux splitting methods generally do not differ significantly[33]. Therefore, the simplest splitting method, known as Lax-Friedrichs will be used.

Again, Equation 3.8 will be used as an example equation to numerically solve. Now if it is possible that $\partial f(u)/\partial u \geq 0$ as well as $\partial f(u)/\partial u \leq 0$, the upwind biasing will need to occur in both directions. This is accomplished through flux splitting. Now for Equation 3.8, f^+ and f^- are denoted where:

$$f(u) = f^+(u) + f^-(u), \tag{3.20}$$

and the requirement is

$$\frac{\partial f^+}{\partial u} \geq 0, \tag{3.21}$$

and

$$\frac{\partial f^-}{\partial u} \leq 0. \tag{3.22}$$

The f^+ and f^- are determined by Lax-Friedrichs flux splitting which is given by:

$$f^{\pm}(u) = \frac{1}{2}(f \pm \alpha u), \tag{3.23}$$

where

$$\alpha \geq \max \left| \frac{\partial f(u)}{\partial u} \right|. \tag{3.24}$$

This splitting can be done *locally* or *globally*, where the free parameter α can be calculated locally, obtaining distinct α values for each grid cell, or a single α can be calculated over the entire grid or declared a constant value over the entire grid in a global splitting. A global splitting is considered to be more robust and will ensure that conservation is not violated while a local splitting will generally be less dissipative[33]. This local splitting is known as *local Lax-Friedrichs* flux splitting as used in Shu and Osher[39]. Wochner suggests a constant global splitting where $300 < \alpha < 400$. This is due to the α value being related to the speed of the waves in the atmospheric medium. Small amplitude waves will travel at the speed of sound, where $c_0 = 343$ m/s. Due to the compressibility of the medium, shock waves at earlier times in the simulation may move faster than 343 m/s, and in local Lax-Friedrichs, α will adapt to each cell in that situation, giving less dissipation. Constant global flux splitting does not adapt at each time step and can easily violate the condition on α. Global flux splitting will adapt, but will cause unnecessary dissipation on areas where information is traveling at contrasting speeds. Local Lax-Friedrichs adds the additional cost of calculating an α for each cell, but avoids the cost of communicating α values over parts of the grid, or the entire grid, and is used mainly for this advantage in the parallel implementation of the model in this work. The Navier-Stokes equations are a generalization of the Euler equations in which viscosity is included. As a result, when calculating α values for each cell in the local Lax-Friedrichs flux splitting, the eigenvalues of the Euler equations can be used to as an approximation of $\partial f(u)/\partial u$ in Equation 3.24. In the x direction, these eigenvalues of the Euler equations are known to be $u + c$ and $u - c$ where u is the x component of velocity as in Equation 2.2 and c is the speed of sound.

The procedure described previously can then be used for f^+ in place of f, which gives $\hat{f}^+_{i+1/2}$ in place of $\hat{f}_{i+1/2}$ in Equation 3.10. For f^- the procedure will be described here. Simply put, f^- will be biased to the right and the combination of stencils will be the exact mirror image with respect to location $x_{i+1/2}$. For f^-, some new definitions are necessary. Instead of Equation 3.10, the numerical flux, $\hat{f}^-_{i+1/2}$, is given as:

$$\hat{f}^-_{i+1/2} = \sum_{q=0}^{k-1} \bar{\bar{\omega}}_q \bar{\bar{f}}^{(q)}_{i+1/2}, \tag{3.25}$$

with

$$\bar{\bar{f}}^{(q)}_{i+1/2} = \sum_{r=0}^{k-1} \bar{c}_{k-1-q,k-1-r} f_{i-q+r+1}, \tag{3.26}$$

and

$$\bar{\bar{\omega}}_q = \frac{\tilde{\tilde{\omega}}_q}{\sum_{l=0}^{k-1} \tilde{\tilde{\omega}}_l}, \tag{3.27}$$

where

$$\tilde{\tilde{\omega}}_s = \frac{\zeta_{k-1-s}}{(\epsilon + \beta_s)^2}, \quad \text{for } s = q, l. \tag{3.28}$$

Along with these new definitions, the previously listed definitions of $\bar{c}$, ζ and β remain. Once $\hat{f}^{+}_{i+1/2}$ and $\hat{f}^{-}_{i+1/2}$ are calculated, they can be recombined by

$$\hat{f}_{i+1/2} = \hat{f}^{+}_{i+1/2} + \hat{f}^{-}_{i+1/2}. \tag{3.29}$$

Finally, this resulting $\hat{f}_{i+1/2}$, calculated at each $i = 1, 2, \ldots, X$ can be used in Equation 3.9 to approximate the right-hand side of Equation 3.8, which in turn, can then be used in the current stage of the Runge-Kutta.

An advantage of the WENO scheme is how easily it extends to multiple dimensions. For example, in a two-dimensional problem such as:

$$\frac{\partial u}{\partial t} = -\frac{\partial f(u)}{\partial x} - \frac{\partial g(u)}{\partial y}, \tag{3.30}$$

the WENO scheme would also be repeated for g in the y direction.

Since the model in this work consists of a set of coupled equations, the WENO scheme is implemented component-wise, meaning it must be applied to each equation of the coupled set. This is done simply by treating $\mathbf{F}$ and $\mathbf{w}$ in the selected model like f and u, respectively, in Equation 3.8 and applying the WENO procedure separately on each equation the components of the vectors describe. Figure 3.1 and 3.2 are graphical depictions of the convex combination of stencils for the third-order WENO scheme ($k = 2$) and the fifth-order WENO scheme ($k = 3$), respectively.

3.3.2 DRP Scheme

The *dispersion-relation preserving* (DRP) scheme was developed by Tam[40] in response to the notion that a high-order method which exhibits consistency, stability, and convergence, does not necessarily mean it will automatically give adequate solutions for wave propagation due to previous schemes being dispersive and dissipative. Therefore, the DRP scheme is manufactured to have the lowest possible dispersion when solving the Euler equations.

Whereas the WENO scheme is generally a high-accuracy method that propagates shocks well, this advantage comes with a large amount of computation. The DRP scheme introduced here is also of high-accuracy (fourth-order), but the amount of computations is less. The disadvantage to this is the ease of which oscillations can be introduced into the simulation, due to shock waves or reflecting boundary conditions, etc. This occurs because the DRP scheme is naive to the state of the medium and calculates the spatial derivatives the same regardless of the function's shape. The DRP scheme uses a single 7-point stencil and is analogous to a standard centered 7-point method for calculating derivatives in how it is applied. To describe the DRP scheme, Equation 3.8 will be used as the example equation to solve numerically, which is restated here as

$$\frac{\partial u}{\partial t} = -\frac{\partial f(u)}{\partial x}. \tag{3.31}$$

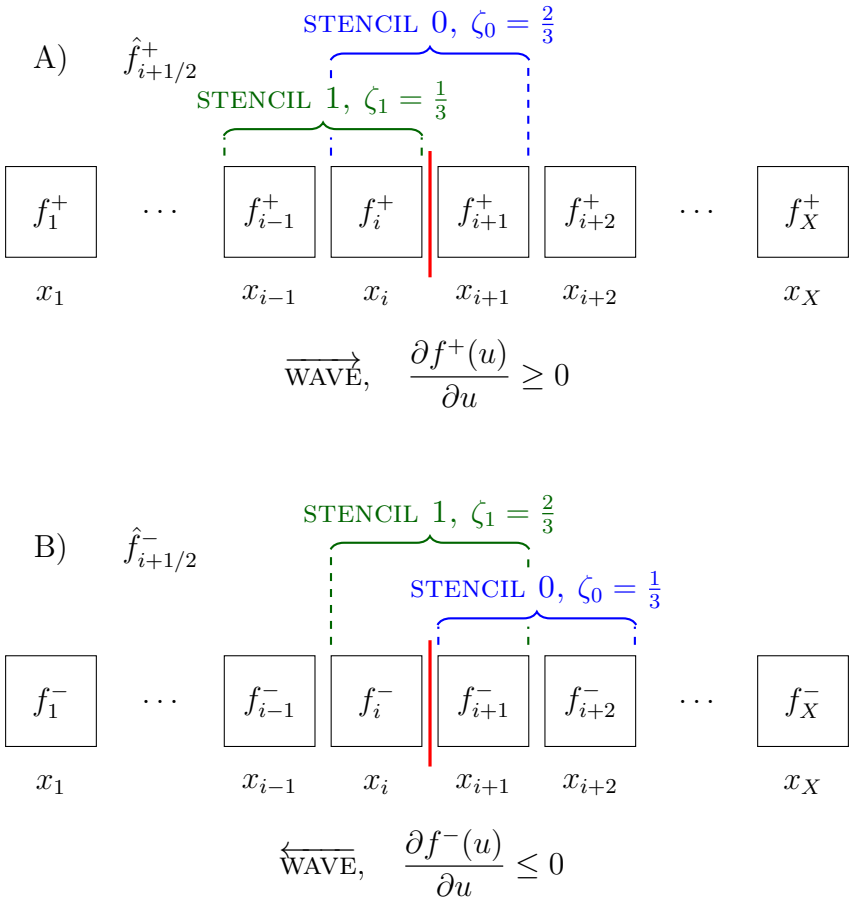

Figure 3.1: A) Stencils used for the calculation of the numerical flux $\hat{f}^+_{i+1/2}$ displayed at the red bar for upwind biased WENO scheme ($k = 2$) for a wave propagating to the right. B) Stencils used for the calculation of the numerical flux $\hat{f}^-_{i+1/2}$ displayed at the red bar for upwind biased WENO scheme ($k = 2$) for a wave propagating to the left.

Using the same computational grid defined previously, the discretized version of Equation 3.31, where $u_i(t)$ is a conservative approximation to $u(x_i, t)$, is now:

$$\frac{du_i(t)}{dt} = -\frac{1}{\Delta x} \sum_{m=-3}^{3} a_m f_{i+m}, \tag{3.32}$$

where

$$\begin{aligned} a_0 &= 0, \\ a_1 &= -a_{-1} = 0.79926643, \\ a_2 &= -a_{-2} = -0.18941314, \\ a_3 &= -a_{-3} = 0.02651995. \end{aligned}$$

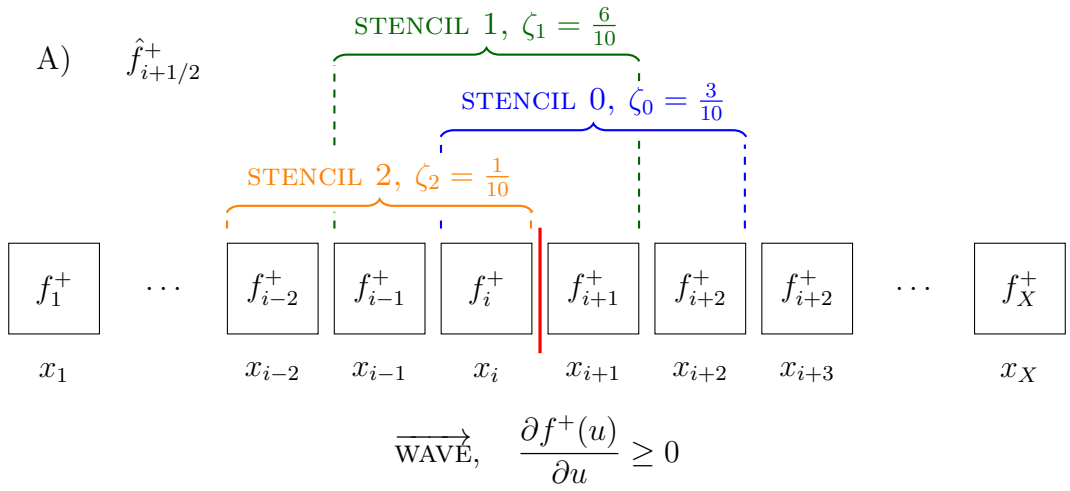

Figure 3.2: A) Stencils used for the calculation of the numerical flux $\hat{f}^+_{i+1/2}$ displayed at the red bar for upwind biased WENO scheme ($k = 3$) for a wave propagating to the right. B) Stencils used for the calculation of the numerical flux $\hat{f}^-_{i+1/2}$ displayed at the red bar for upwind biased WENO scheme ($k = 3$) for a wave propagating to the left.

Using a computational grid of $X \times Y$ cells, the DRP scheme is applied to Equation 3.1 as:

$$\frac{\partial \mathbf{w}_{i,j}}{\partial t} = -\frac{1}{\Delta x}\sum_{m=-3}^{3} a_m \mathbf{F}_{i+m,j} - \frac{1}{\Delta y}\sum_{m=-3}^{3} a_m \mathbf{G}_{i,j+m} + \mathbf{H}_{i,j}, \tag{3.33}$$

using a_m as defined previously.[1] Figure 3.3 depicts the DRP stencil on a grid.

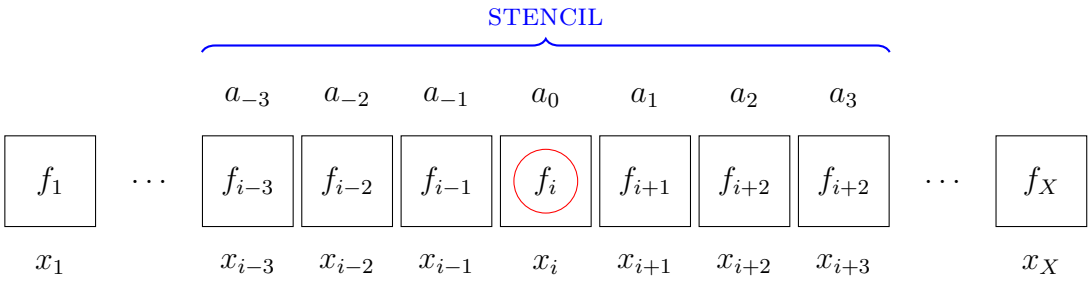

Figure 3.3: The stencil used in the DRP scheme to calculate $\partial f(u)/\partial x$ at the location x_i.

3.3.3 Hybrid Scheme

As stated previously, the WENO scheme is capable of capturing shock waves. Therefore, it is advantageous to apply the WENO scheme to areas containing shocks. However, the WENO scheme is known to have moderate numerical dispersion and more calculations are required per grid cell than the DRP scheme. However, the DRP scheme should not be used on regions of the domain containing discontinuities, because non-physical oscillations will occur. The non-physical oscillations possible in the DRP scheme are known to not decay even when the grid is refined[41]. Therefore it is desirable to use a hybrid of both of these schemes to exploit the advantages of each of them. To do this, a procedure must be used to make the decision on whether to use the WENO scheme or the DRP scheme for a particular region. Although this procedure adds extra computations not necessary when using only a single scheme, the computational savings of using a hybrid of them far outweigh the additional computations. A decision must also be made on the granularity in which the schemes can switch between one another. In some hybrid schemes, the regions where the differing schemes meet are overlapped and a weighted average between their results is used over those regions. In this work, the schemes will have the ability to be switched on a per

[1] Note the bold-face vectors are discretized grids on a standard x, y plane in this case and not matrices. Therefore the notation for the subscripts of the bold-face vectors indicate indices into the discretized grid and not indices into a matrix. This distinction is necessary as grids and matrices have differing points of entry and different mappings of indices into themselves.

cell basis. This is advantageous due to the schemes not overlapping each other, which produces no necessary extra computations. It is disadvantageous due to the schemes possibly giving slightly different values at adjacent grids that would not necessarily occur when using a single scheme. In a sufficiently rough region, this could introduce oscillations into the DRP scheme regions. However, this can be avoided simply by setting a lower tolerance for which the WENO scheme should be activated for particular cells. Therefore, the border at which the WENO and DRP schemes meet should appear only around the smooth regions near ambient values where waves do not exist, promoting less discrepancy between neighboring cell values. The reader is referred to Ren[41], Kim and Kwon[42], Shen and Yang[43], Abdalla[44], and Pirozzoli[45] for more information on hybrid schemes where the WENO scheme is used as one of the subschemes.

The hybrid scheme is almost entirely based on how to choose between the two subschemes. To allow a decision between schemes, a smoothness indicator that is bounded is introduced[2]. The smoothness indicator used is a modified version from the version used in Ren[41] and Kim and Kwon[42], and is given by:

$$r_i = \frac{2\left|\Delta f_{i+1/2}\Delta f_{i-1/2}\right| + \epsilon}{(\Delta f_{i+1/2})^2 + (\Delta f_{i-1/2})^2 + \epsilon}, \tag{3.34}$$

where

$$\Delta f_{i+1/2} = f_{i+1} - f_i. \tag{3.35}$$

The $\epsilon > 0$ is used to avoid division by zero. It can be seen that $r_i \in [0, 1]$. This allows a threshold value $r_c \in [0, 1]$ to be chosen which dictates what is considered smooth or not. The ϵ is defined as

$$\epsilon = \frac{0.9r_c}{1 - 0.9r_c}\xi^2. \tag{3.36}$$

where $\xi > 0$ is user-defined. In this work, $\xi = 1$ and the value of the threshold r_c is mainly based on trial and error since there is no general method for choosing it. As values of r_c decrease toward zero, this indicates the function is more likely to be considered not smooth, which in this case would mean the WENO algorithm would be given more widespread use. In this work, r_c will be used solely as the threshold value, although it is possible to use ξ as a threshold value as well. A plot of how ϵ changes with respect to r_c can be seen in Figure 3.4. The smoothness indicator can be calculated for neighboring cells as well, allowing the smoothness indicator to inspect a designated size area around the current cell. This is useful for ensuring that the WENO scheme will be used on cells nearby discontinuities as well.

For the hybrid scheme, it may be helpful to recognize how both the WENO and DRP schemes calculate the value for any given cell x_i. To calculate $\partial f(u)/\partial x$, the WENO scheme must calculate the fluxes at the faces of the cell x_i, which implies

[2]It should be noted that the smoothness indicators in the WENO scheme cannot be used because they are not calculated if the WENO scheme is not being used in a particular region.

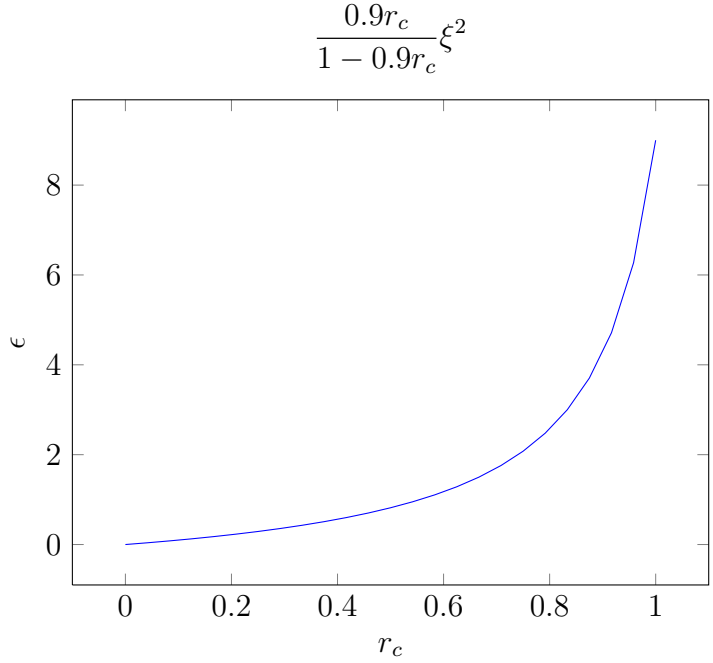

Figure 3.4: Plot of ϵ and r_c with $\xi = 1$.

the WENO scheme must be applied twice for each cell, giving $\hat{f}_{i+1/2}$ and $\hat{f}_{i-1/2}$, then $(\hat{f}_{i+1/2} - \hat{f}_{i-1/2})/\Delta x$ must be calculated to arrive at a value centered on x_i. The DRP scheme calculates the value for $\partial f(u)/\partial x$ at x_i directly. Figure 3.5 gives a visual representation of how these two schemes fit together in the hybrid scheme.

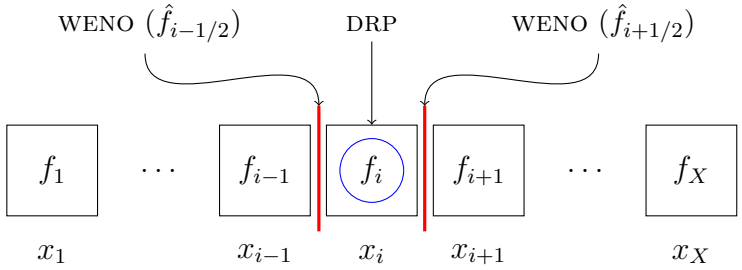

Figure 3.5: A graphical depiction of how the hybrid scheme calculates $\partial f(u)/\partial x$ at the location x_i using either the WENO or DRP subscheme. After the conclusion of the WENO scheme, $(\hat{f}_{i+1/2} - \hat{f}_{i-1/2})/\Delta x$ must be calculated to arrive at a value centered on x_i, while the DRP scheme is intrinsically centered on x_i.

3.4 Stability

Both the WENO and DRP schemes are *explicit* methods, meaning that they are conditionally stable. Requirements must be met to ensure that each method is able to produce legitimate results. This is done through a parameter known as the *Courant-Friedrichs-Levy condition*, or CFL condition. This condition is used to impose a maximum time step size which will yield stable results corresponding to the chosen grid cell size. The maximum time step is calculated by:

$$\Delta t = \text{CFL}\frac{\Delta x}{c}, \tag{3.37}$$

where, for stability in the DRP scheme the condition to be met is that CFL < 1, while in the WENO scheme the condition is that CFL < 0.5. Therefore, for the hybrid scheme it is also the requirement that the CFL < 0.5 for stability. The term $\Delta x/c$ denotes the propagation time required to reach a distance of Δx. It can be seen from Figure 3.5 that the WENO scheme uses the two fluxes at the cell faces, while the DRP scheme calculates one value at the center of the cell. This is the reason why the WENO scheme must allow a propagation time over $2\Delta x$ (CFL < 0.5) while the DRP scheme only needs a propagation time over Δx (CFL < 1).

As stated earlier, molecular relaxation also has an effect on the maximum time step size of the model if it is to be included. When including molecular relaxation for nitrogen, the requirement is that

$$\Delta t \leq \frac{\tau_{N_2}}{4}. \tag{3.38}$$

If molecular relaxation due to oxygen is included, the requirement is that

$$\Delta t \leq \frac{\tau_{O_2}}{4}. \tag{3.39}$$

This condition is necessary regardless of whether or not molecular relaxation due to nitrogen is included into the model or not. If this condition is not met, then the model will be unstable. If molecular relaxation is included in the model and these conditions are met as well as the CFL condition, then the model will be stable.[3]

When calculating τ_{N_2} and τ_{O_2} and the resulting Δt, another reason can be seen why molecular relaxation due to oxygen is excluded in this work. With $T_{ref} = T_0 = 293.16$ K, $p_{ref} = p_0 = 101325$ Pa and $RH = 20$, this gives relaxation frequencies of $f_{N_2} = 138.24$ Hz and $f_{O_2} = 10556.99$ Hz, corresponding to relaxation times of $\tau_{N_2} = 1.1512 \times 10^{-3}$ s and $\tau_{O_2} = 1.5075 \times 10^{-5}$ s. Therefore,

$$\Delta t \leq \frac{\tau_{O_2}}{4} = 3.7689 \times 10^{-6} < \frac{\tau_{N_2}}{4} = 2.8782 \times 10^{-4}, \tag{3.40}$$

[3]This is assuming the source inserted into the model is sufficiently smooth to promote stability, which will be discussed further in Chapter 4.

which, if oxygen was included, would be a prohibitively small time step for the large size domains and simulation times which are a goal of this work.

In this work, the speed of sound c, varies with altitude in that it decreases as altitude increases. Due to this aspect, the speed of sound at the ground level should be used in the stability condition. This would mean the largest value for c should be used, which in turn, gives the lower bound on the maximum time step size.

3.5 Summary

In this chapter, the methods for solving the model equations have been detailed. Either a third-order or second-order Runge-Kutta scheme will be used for the time derivative, while a hybrid scheme utilizing the WENO and DRP subschemes is used for the spatial derivative. The WENO scheme will be used over areas of the domain in which high-amplitude waves along with possible discontinuous waves are propagating while the DRP scheme will be selected for use in areas of the domain that propagating waves are sufficiently smooth or do not exist. The hybrid scheme allows for significant computational cost savings over much of the simulation while still exhibiting the ability to stably propagate shock waves. A smoothness indicator for choosing between subschemes on a per cell basis was also detailed. Although, the smoothness indicator calculation would not be necessary when implementing either subscheme on its own, this added calculation is minor when compared to time the computer will spend in calculations for the WENO scheme or the DRP scheme. The stability of each scheme was also discussed while noting that the small time steps imposed due to the inclusion of molecular relaxation due to oxygen are small enough to be prohibitive when dealing with the large size domains and simulation times in this work and is therefore another reason why molecular relaxation due to oxygen is excluded from the model.

Chapter 4

Implementation

4.1 Introduction

With the model and the chosen methods for solving the model equations having been described, this chapter focuses on the actual implementation of the model and methods in the computer, as well as the issues associated with it. The first section describes how a lightning channel source is to be fabricated and inserted into the model using a specific algorithm newly designed for the purpose. The algorithm is designed to draw the lightning channel using a few primitive mathematical functions, and with the ability to draw the source in a sufficiently smooth manner to ensure that the sources will promote stability. The implementation of simple boundary conditions for the model is then discussed which will be the last issue remaining to be discussed before being able to write a program for the model. This leads into an outlining of the pseudocode of a static mesh version of the program. The last issue that is dealt with is implementation of the technique of *adaptive mesh refinement*[1] (AMR) which will be key in allowing the large range of scale in the simulation and the larger domain sizes required in this work. A single person attempting to program AMR functionality into their model unaided in some higher level form is nearly an intractable problem.[2] Therefore a C++ library to aid with the implementation of AMR will be utilized. A short background on AMR and an overview of available software libraries for implementing AMR is given, while the implementation of AMR in the computer will be aided by the use of a C++ library framework known as SAMRAI.

[1] Note that in this chapter, the use of the words "grid" and "mesh" may be used interchangeably, but will have the same meaning in this case.

[2] Manazares[46] has demonstrated very detailed work on the direct implementation of AMR into a model that is very applicable to the model and methods in this work.

4.2 Source Fabrication

For the model to simulate anything but an ambient system, a source (or sources) must be inserted. This is done through the manipulation of the grid values associated with the first component of $\mathbf{H}$, denoted as $\mathbf{H}(1)$.[3] Using the first component of $\mathbf{H}$ indicates that the source being inserted will be a *density* source. The model is general enough that conceivably any shape source could be inserted into the model provided it is smooth enough everywhere to promote stability. When inserting a source, it will be easiest if the source can be described as a purely mathematical function, as that function can then be used on the right-hand side of Equation 3.1 directly to draw the source. However, it is also possible to draw on the source grid algorithmically, or even by superimposing pixel values from an image onto the grid. By using an image such as a bitmap to draw lightning channels onto the source grid, a user could fashion input quite easily to their liking and have total control over what the source should look like, even by processing actual images of lightning for input. The drawback to that approach is that the input image would have to be the same resolution as the discretized grid in the problem.[4] This would lead to huge file sizes on the order of gigabytes. Therefore, to have an effective process for drawing lightning channels, while avoiding having to deal with such a large amount of input data, an algorithmic approach will be introduced that fuses together a set of primitive functions which can be controlled mathematically. This approach will reduce the source input file size to a list of vertices that describe the lightning channel geometry while still allowing for a great deal of control over the properties of the source channel.

4.2.1 Generating the Lightning Channel

The first step in drawing a complete lightning channel on the source grid will be to generate a list of vertices describing the start point and end point of each segment in the entire channel. This list can then be read into the simulation and drawn on the source grid using Gaussian-distributed segments which will be discussed in Section 4.2.3.

For the reason that lightning channel geometries can be wildly random, this work is more focused on the numerical modeling of the resulting waves of a lightning strike rather than attempting to progress the methods for creation of lightning channel geometries. In Kim and Lin[47][48] and Matsuyama *et al.*[49], methods for the composition, animation and rendering of spark discharges and lightning channels have been accomplished by solving a Laplace equation, resulting in an establishment of an electrical field which then creates the discharge pattern geometry. Although this is

[3]Note that the indices of vectors may change from beginning at 1 or 0 depending on the programming language of focus in a particular section.

[4]Note the image should have to be at least the resolution of the y dimension, but not necessarily the x dimension as the lightning channel would not necessarily span the entire x dimension.

a more physically-based approach, their method involves discretizing a domain and using a numerical method to solve the Laplace equation which makes creating the lightning geometry very computational on its own and not ideal for producing source channels at arbitrary resolutions as will be necessary when moving the model into an AMR setting which will be discussed in Section 4.5. Therefore their method is not preferred for the size domains in this work and will not be investigated further. The process that is used for generating a lightning geometry is a simple random walk much like the ones used in Ribner and Roy[6] and Glassner[9]. The result of the process will be a list of segments that will comprise a random lightning channel geometry.

The angles between sequential segments and the length of those segments are the main pieces of data necessary to construct the lightning channel geometry. According to Hill[50] the mean absolute value of angles between segments studied between 5 and 70 meters in length in a lightning channel are about 16 degrees. In this work, creation of a lightning channel is accomplished in a fashion relatively close to the manner in which Ribner and Roy created them. However, they were only concerned with the construction of the main channel of a lightning strike and did not construct channels that were allowed to branch. In this section their work is extended to allow branching in the lightning channel.

To begin the fabrication of the lightning channel, they suggest a probability density function for choosing the angle for each segment, θ_n, of the form:

$$P(\theta_n - \bar{\theta}_{n-1}) = \exp\left(-\frac{\theta_n - \bar{\theta}_{n-1} + \frac{\bar{\theta}_{n-1}}{N}}{\Delta\theta_m^2}\right), \tag{4.1}$$

where

$$\bar{\theta}_{n-1} = \frac{1}{k}\sum_{n-1-k}^{n-1}\theta_i.$$

In these equations, θ_n is the randomly chosen angle for the current segment, N is a biasing term toward the vertical to keep the channel from straying too far in both directions horizontally, $\Delta\theta_m$ is the mean absolute angle between segments, and k is the amount of previous angles adjacent to the current angle to average and bias toward. Ribner and Roy call this technique *memory smoothing*. Ribner and Roy use values $\Delta\theta_m = 30$, $N = 20$, and $k = 4$ which are used here and have been shown to give convincing results. For the segment length Ribner and Roy suggest a poisson distribution and Glassner gives a polynomial probability density function that changes according to eight different categories of height of the segment. For simplicity, in this work, segment lengths along the channel are fixed to 6 meters. The possibility of branching Ribner and Roy did not allow for will be added by using a recursive algorithm that changes the pertinent parameters at each level of branching. Example pseudocode for the lightning channel geometry is given in Algorithm 4.1 while sample MATLAB code for this algorithm can be seen in Appendix A. A visual aid is given in Figure 4.1.

Algorithm 4.1: Pseudocode for generating a random lightning channel as a list of segments.

```
1  LIGHTNINGCHANNEL(domainW, domainH, startX, startY, maxIterations,
2    branchLevels, meanAngle, N, branchyness, meanBranchLength,
3    numAvgAngles)
4  x(1) ← startX; y(1) ← startY; i ← 0;
5  while y(1) > 0 and i < maxIterations do
6    θ ← GETANGLE(meanAngle, angles, N); length ← GETLENGTH();
7    for n ← (numAvgAngles − 1) to 1 do
8      angles(n + 1) ← angles(n);
9    end
10   angles(1) ← θ;
11   x(2) ← x(1) + length · cos(90° − θ); y(2) ← y(1) + length · sin(90° − θ);
12   if y(2) < 0 then
13     y(2) ← 0;
14   end
15   Write segment x(1), y(1), x(2), y(2) to file;
16   x(1) ← x(2); y(1) ← y(2); i ← (i + 1);
17   if branchLevels > 0 then
18     if rand() < branchyness then
19       branchMaxIterations ← maxIterations/5;
20       LIGHTNINGCHANNEL(domainX, domainY, x(2), y(2),
21         branchMaxIterations, branchLevels − 1, meanAngle, N,
22         branchyness/10, meanBranchLength/2, numAvgAngles);
23     end
24   end
25 end
```

In Algorithm 4.1, *domainW* and *domainY* are the width and height of the domain, respectively. The coordinates of the start of the lightning channel are put into *startX* and *startY*. The parameter *maxIterations* is used for two purposes. The first is as a failsafe to limit the amount of iterations the main loop will execute in case the channel happens to stray too far horizontally. N also helps avoid that situation. The second is when the algorithm calls itself and branches, this parameter dictates how long the branched channel will be. The *branchLevels* parameter indicates how many recursive branches the lightning channel will be composed of. The *meanAngle* parameter is the average angle that θ_n should be relative to the average of the k previous segments. N is the biasing parameter toward the vertical. The *branchyness* parameter dictates how likely a branch would be formed at each iteration of the main loop. The *meanBranchLength* parameter is the average length of the entire

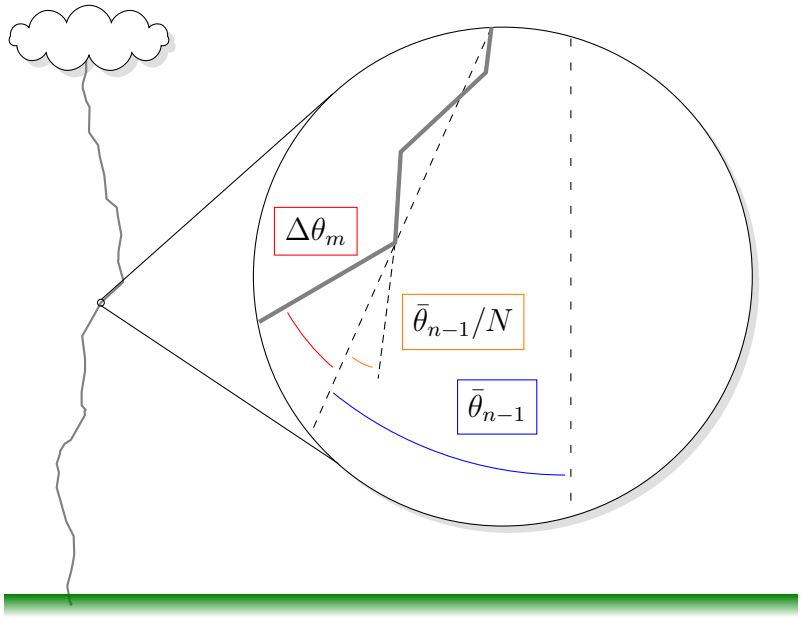

Figure 4.1: The fabrication of a lightning channel geometry. The blue arc is the angle from the vertical that is an average of the last k segment angles. The red arc is the mean angle deviation from the average of the k previous angles. Lastly, the orange arc is the term that introduces a bias in the random walk toward the vertical.

branches at the first branch level. Finally, *numAvgAngles* (k in Equation 4.1) tells the algorithm how many previous angles should be averaged to bias the next segment angle relative to that average.

In summary, Algorithm 4.1 takes these parameters as input and iterates until a segment touches the ground or reaches a certain amount of iterations. It decides on a random angle for the next segment based on the probability density function (PDF) given in Equation 4.1. Next, a length for the segment is chosen (6 m in this case, but could be determined by a different PDF). Each element of the array of angles to average, *angles*, is then shifted downward and the last angle discarded, while the newest chosen angle, θ_n, is put into the first location of *angles*. Next, the coordinates of the end of the segment are calculated at lines 11–12 and stored.[5] Then the coordinates of the beginning and ending of the segment are written to a file and the coordinates of the end of the segment are then stored as the coordinates of the beginning of the next segment. Finally, if more than zero branch levels are desired, a random number between 0 and 1 chosen from a uniform distribution, is generated and if *branchyness* is less than that number, the algorithm recursively calls itself. The parameters used when calling LIGHTNINGCHANNEL again are modified to use the end of the current segment as the beginning of the next, where *branchMaxIterations* is recalculated

[5]The sin and cos functions are taking degrees, not radians, as input in this case.

to shorten each recursive branch by a factor of 5, *branchLevels* is decremented by 1, *branchyness* is decreased by a factor of 10 and *meanBranchLength* is decreased by a factor of 2. These factors give reasonable results, but they may be adjusted for other desired results.

4.2.2 The Primitive Drawing Functions

Once the lightning channel geometry is defined as a list of segments, it must then be drawn by another algorithm onto the source grid. As stated earlier, the source must be sufficiently smooth everywhere for the model to be stable. This is accomplished by fusing together two primitive Gaussian-distributed functions: the *point* and the *line*.

4.2.2.1 Gaussian-Distributed Points

A Gaussian-distributed or smoothed point at a point (x_0, y_0) can be described in two dimensions by the function:

$$f(x,y) = A \cdot \exp\left(-\frac{\ln(2)}{\alpha_s^2}((x-x_0)^2 + (y-y_0)^2)\right). \tag{4.2}$$

Here, A is the amplitude of the point and α_s is the Gaussian half-width which is used to narrow or widen the distribution of the point. An example of a Gaussian-distributed point can be seen in Figure 4.2.

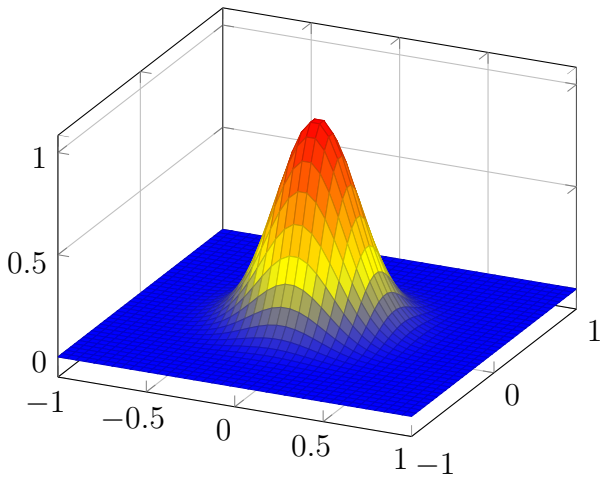

Figure 4.2: Plot of a Gaussian-distributed point.

4.2.2.2 Gaussian-Distributed Lines

A Gaussian-distributed or smoothed line through a point (x_0, y_0) and rotated by an angle θ can be described in two dimensions by the function:

$$f(x,y) = A \cdot \exp\left(-\frac{\ln(2)}{\alpha_s^2}(\sin(\theta)(x-x_0) + \cos(\theta)(y-y_0))^2)\right). \tag{4.3}$$

Again, A is the amplitude of the point and α_s is the Gaussian half-width which is used to narrow or widen the distribution of the line. An example of a Gaussian-distributed line can be seen in Figure 4.3.

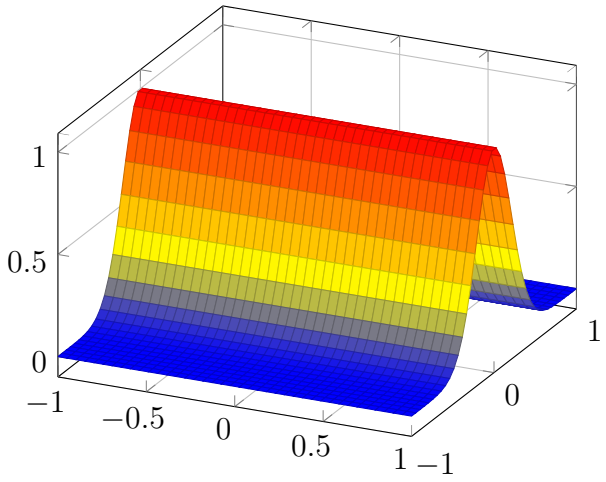

Figure 4.3: Plot of a Gaussian-distributed line.

4.2.3 Drawing the Lightning Channel

Using both the point and line primitive functions, a Gaussian-distributed segment can be drawn. The idea of drawing a segment between two points (x_0, y_0) and (x_1, y_1) on a grid is to first draw a Gaussian-distributed line strictly between the two points using Equation 4.3. Then Equation 4.2 is used to draw half of a Gaussian-distributed point at each end of the truncated line, so as to "cap" each end of it. This is necessary to maintain smoothness everywhere on the segment. An example of a Gaussian-distributed line segment can be seen in Figure 4.4.

The process of "capping" the ends of the truncated line would be trivial if the line was perfectly vertical or horizontal on the grid because the algorithm could stop drawing when it gets past both points, strictly in the x or y direction. However, the segment must possess the ability to be rotated at any angle, since the end points of the segments on the grid are arbitrary. If the segment is rotated at some arbitrary

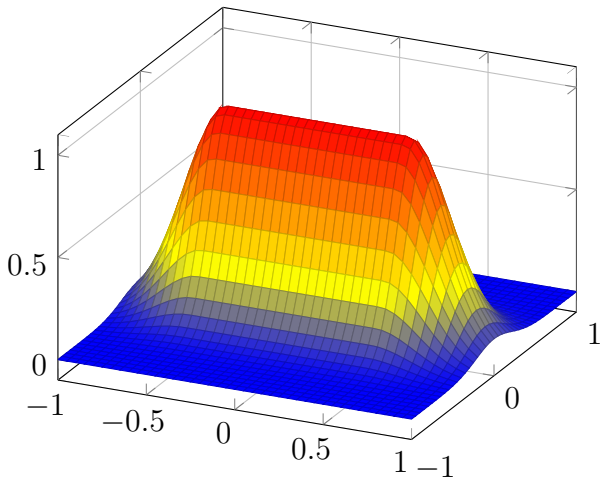

Figure 4.4: Plot of a Gaussian-distributed segment.

angle, it needs to be aware of the barrier at the ends of the line to switch from drawing a point to drawing a line and vice versa. This barrier would be perpendicular to the line itself. A visual representation of this idea can be seen in Figure 4.5.

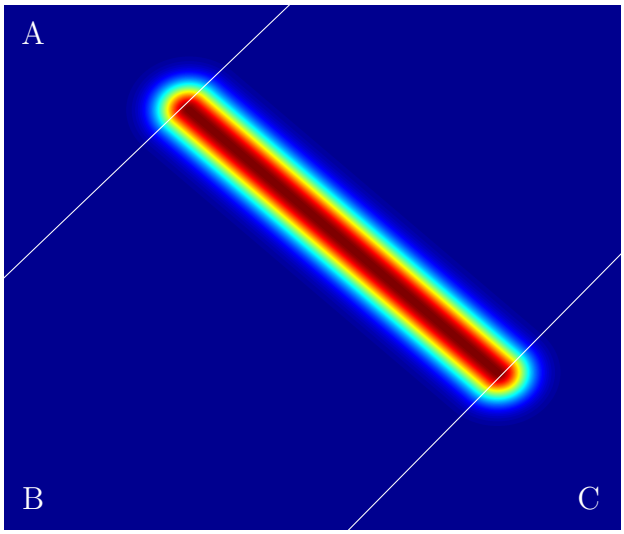

Figure 4.5: A visual depiction of how the primitive drawing functions are fused together to create a Gaussian-distributed segment. In sections A and C, the algorithm will draw using a Gaussian-distributed point, while in section B, it will draw using a Gaussian-distributed line.

Once a Gaussian-distributed segment can be drawn between any two points on the grid, they must be able to be connected together while maintaining smoothness. When iterating through the list of vertices of the segments and drawing the segments, the entirety of cells in the source grid will be iterated through for each segment, so care must be taken due to overlapping source amplitude values at connected vertices. A naive approach of just adding source amplitude values to each grid cell will create inconsistent values at the connected vertices which would effectively be double the actual amplitude desired. Therefore, the approach taken should be done in the following manner:

$$\mathbf{H}(1) = \mathbf{H}(1) + F(x, y) \cdot \left(1 - \frac{\mathbf{H}(1)}{A}\right). \tag{4.4}$$

In Equation 4.4, $\mathbf{H}(1)$ is the density source grid, $F(x, y)$ is the value of whatever primitive function is currently being drawn on the segment (half point cap on one side, line, or half point cap on other side), and A is the source amplitude. Adding each segment to the grid in this manner has been shown to result in a lightning channel geometry source that is sufficiently smooth everywhere provided the Gaussian half width for the primitive functions is sufficiently wide. Pseudocode for the algorithm can be seen in Algorithm 4.2–4.3. Figure 4.6 shows two overlapping segments drawn on a grid while Figure 4.7 shows an entire lightning channel source grid drawn using Algorithm 4.2–4.3.[6]

4.3 Boundary Conditions

Since the domain for the problem is defined finitely within a rectangular shape, physical boundary conditions for the domain must be enacted at each edge of the domain to maintain versimilitude. To simulate an open-ended atmosphere, the left, right and top boundaries should allow the waves to propagate out of the domain. Another approach which would give a similar result would be to have the boundaries absorb the waves. Allowing the waves to exit the domain is not a trivial task however. Although there are techniques for doing so such as using a *perfectly matched layer* as proposed in Chapter 6, absorbing boundary conditions are the simplest option for the task. Therefore, absorbing boundaries are used on the left, right, and top edges of the domain. The ground is rigid and would, for that reason, reflect propagating waves from it in the opposite direction. Hence, a reflecting boundary is used at the bottom edge of the domain. It can be seen in both the fifth-order WENO and fourth-order DRP schemes that at a particular grid cell location (i, j), grid cells from $i-3$ to $i+3$ and $j-3$ to $j+3$ are referenced in the calculation. To handle this behavior at the domain boundary, a 3-cell perimeter around the grid is introduced and will be known as the

[6] In reality, branches of lightning channels should not necessarily be the same amplitude as the main channel, but this feature has not been incorporated into the algorithm.

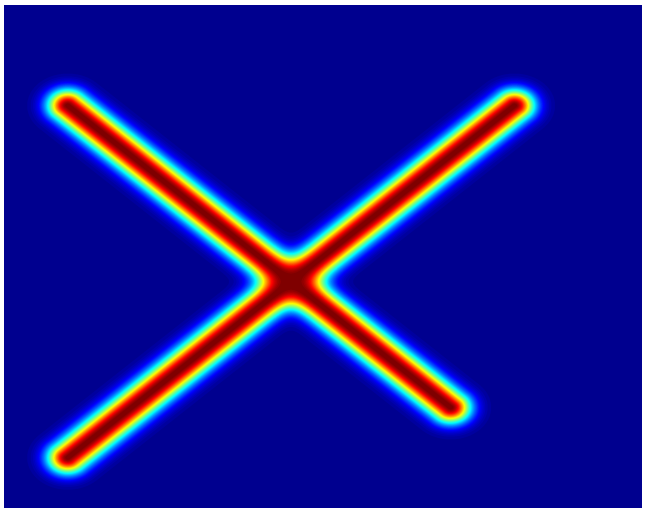

Figure 4.6: An example of two overlapping segments drawn sequentially using Algorithm 4.2–4.3 which takes advantage of Equation 4.4. If the segments were to be drawn naively through only addition, the overlapping areas of the segments would be at twice the amplitude.

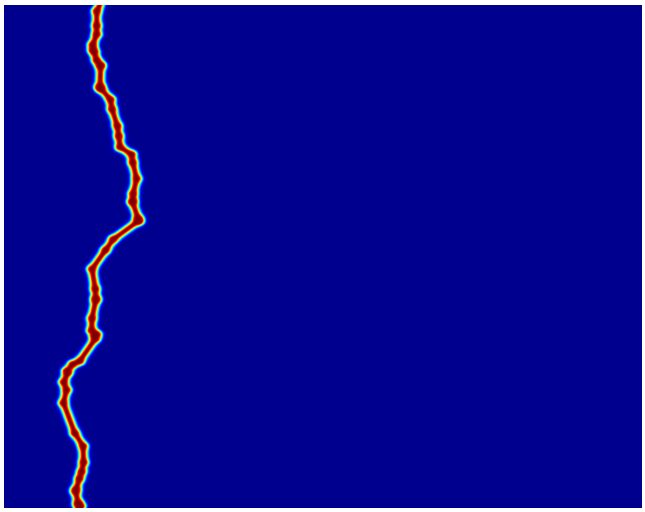

Figure 4.7: An example source drawn on the source grid using the lightning channel algorithm.

Algorithm 4.2: Pseudocode for drawing a Gaussian-distributed segment. (Continued in Algorithm 4.3)

```
1  drawSource(domainW, domainH, X, Y, α_s, A)
2  dx ← domainW / X; dy ← domainH / Y;
3  Read list of vertices for n segments from file into S_{4×n};
4  for k ← 1 to n do
5      θ ← − arctan( (S_{4,k} − S_{2,k}) / (S_{3,k} − S_{1,k}) );
6      if (S_{1,k} > S_{3,k} and S_{2,k} > S_{4,k}) or
7         (S_{1,k} ≤ S_{3,k} and S_{2,k} ≥ S_{4,k}) then
8          orientation ← 1;
9      else if (S_{1,k} ≥ S_{3,k} and S_{2,k} ≤ S_{4,k}) or
10        (S_{1,k} < S_{3,k} and S_{2,k} < S_{4,k}) then
11         orientation ← 2;
12     end
13     for j ← 1 to Y do
14         yw ← dy · (j − 1/2);
15         for i ← 1 to X do
16             xw ← dx · (i − 1/2);
17             line0 ← tan(π/2 − θ) · (xw − S_{1,k}) + S_{2,k};
18             line1 ← tan(π/2 − θ) · (xw − S_{3,k}) + S_{4,k};
19             if (orientation = 1 and yw ≥ line0) or
20                (orientation = 2 and yw ≤ line0) then
21                 F ← A · exp( −(ln(2)/α_s^2) · ((xw − S_{1,k})^2 + yw − S_{2,k})^2) );
22             end
```

ghost cells. They are called this because they are necessary for calculation, but do not formally exist in the domain. Manipulation of these ghost cells will be involved when imposing boundary conditions. The boundary conditions are to be imposed at each stage of the Runge-Kutta.

There is another boundary condition that will not be used directly in this work, but yet it has a particular advantage nonetheless. This boundary condition is known as a *periodic* boundary condition in which a dimension is chosen to virtually connect both ends of the domain together. The advantage to this is that it allows for unbounded propagation to occur in that dimension while confining calculations to a small bounded domain. Using a periodic boundary naively on the left and right boundaries would allow both left traveling and right traveling waves to collide with

Algorithm 4.3: Pseudocode for drawing a Gaussian-distributed segment. (Continued from Algorithm 4.2)

```
        if (orientation = 1 and yw ≤ line1) or
          (orientation = 2 and yw ≥ line1) then
            F ← A · exp( −(ln(2)/α_s^2) · ((xw − S_{3,k})^2 + yw − S_{4,k})^2) );
        end
        if (orientation = 1 and yw < line0 and yw > line1) or
          (orientation = 2 and yw > line0 and yw < line1) then
            F ← A·
              exp( −(ln(2)/α_s^2) · (sin(θ)(xw − S_{1,k}) + cos(θ)(yw − S_{2,k}))^2 );
        end
        H1_{i,j} ← H1_{i,j} + F · ( 1 − H1_{i,j}/A );
      end
    end
end
```

one another in an unnatural manner. Therefore a temporary absorbing boundary could be enacted on the left boundary which would serve to absorb the left traveling waves and after it has sufficiently absorbed all left traveling waves, it could then be shut off and the simulation allowed to propagate right traveling waves any desired distance along the x dimension.

4.3.1 Reflecting Boundary Condition

Although the ground is rigid to an extent, it is not perfectly rigid. However, implementing perfectly rigid boundary conditions is much simpler than boundary conditions which partially absorb waves at a desired rate first and then reflect them (a combination of multiple boundary conditions is typically known as a Robin condition). It is for this reason that a perfectly reflecting boundary condition is used for simulating the ground.

Imposing the reflecting boundary condition is done by:

$$\mathbf{w}_{i,j} \leftarrow \mathbf{w}_{i,4}, \forall i \text{ and for } j = 1, 2, 3, \tag{4.5}$$

and then

$$\mathbf{w}(3)_{i,3} \leftarrow -\mathbf{w}(3)_{i,3}, \forall i. \tag{4.6}$$

This effectively copies the data from the bottom edge of the domain to the ghost cells on the bottom edge and then reverses the direction of the v component of the wave

velocity on the ghost cells next to the edge, simulating a wave's interaction with a rigid material. Although this reflecting boundary is defined for the ground edge of the domain, this can be applied at the other boundary edges if desired.

4.3.2 Absorbing Boundary Condition

An ideal boundary condition for the top, left and right boundaries of the domain would be one that lets the propagating waves freely exit the domain. However, implementing such boundaries is not a straightforward task. The simplest boundaries to enact that give a similar property are absorbing boundary conditions. The disadvantage to using simple absorbing boundary conditions is that they do not necessarily absorb the waves completely and will reflect a small amount of the waves back into the domain. Although this consequence is not easily avoidable, it can usually be made small enough to be tolerated.

Imposing the absorbing boundary condition at an edge is done through the construction of a Gaussian envelope, denoted E_{top} for the top edge of the domain for example. An example of the envelope which would be used on the left edge of the domain, E_{left} is depicted in Figure 4.8. This envelope is then used to attenuate the

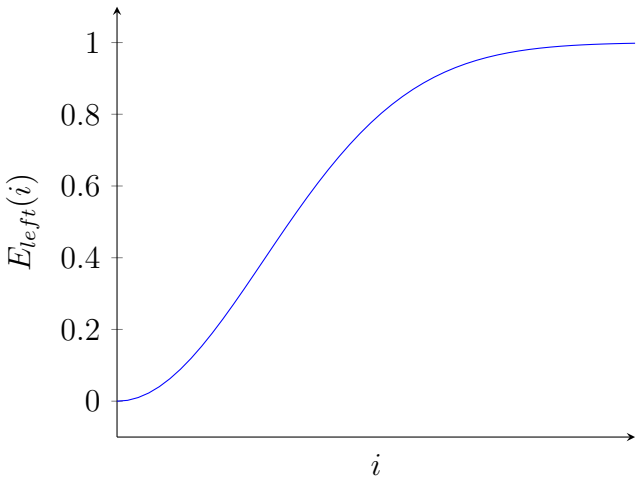

Figure 4.8: A plot of the absorbing Gaussian envelope used to impose an absorbing boundary condition at the left edge of the domain.

waves as they approach the domain edge. Mathematically this is done by shifting the variables by their initial conditions, multiplying the variables by the Gaussian envelope, and then shifting them back by their initial conditions. An example of doing

this at the top edge of the domain using E_{top} in the full model is as follows:

$$\begin{aligned}
\mathbf{w}(1)_{i,j} &= (\mathbf{w}(1)_{i,j} - \rho_0) \cdot E_{top}(j) + \rho_0, \quad \forall i \text{ and for } j = Y - n, \ldots, Y, \\
\mathbf{w}(2)_{i,j} &= (\mathbf{w}(2)_{i,j}) \cdot E_{top}(j), \quad \forall i \text{ and for } j = Y - n, \ldots, Y, \\
\mathbf{w}(3)_{i,j} &= (\mathbf{w}(3)_{i,j}) \cdot E_{top}(j), \quad \forall i \text{ and for } j = Y - n, \ldots, Y, \\
\mathbf{w}(4)_{i,j} &= (\mathbf{w}(4)_{i,j}) \cdot E_{top}(j), \quad \forall i \text{ and for } j = Y - n, \ldots, Y, \\
\mathbf{w}(5)_{i,j} &= (\mathbf{w}(5)_{i,j} - \rho_0 \cdot T_0) \cdot E_{top}(j) + \rho_0 \cdot T_0, \quad \forall i \text{ and for } j = Y - n, \ldots, Y, \\
\mathbf{w}(6)_{i,j} &= (\mathbf{w}(6)_{i,j} - \rho_0 \cdot T_0) \cdot E_{top}(j) + \rho_0 \cdot T_0, \quad \forall i \text{ and for } j = Y - n, \ldots, Y, \\
p_{i,j} &= (p_{i,j} - p_0) \cdot E_{top}(j) + p_0, \quad \forall i \text{ and for } j = Y - n, \ldots, Y, \\
T_{i,j} &= (T_{i,j} - T_0) \cdot E_{top}(j) + T_0, \quad \forall i \text{ and for } j = Y - n, \ldots, Y,
\end{aligned}$$

where

$$E_{top}(j) = \exp\left(-\frac{\ln(2)}{\alpha_e^2} \cdot (Y - j - y_s)^2\right) + 1. \tag{4.7}$$

Likewise, this is done at the left and right domain boundaries as well using similar Gaussian envelopes designed for each edge. In the previous list of equations relating to Equation 4.7, n is the number of cells from the border in which to apply the absorption. This allows the computer to avoid iterating through the entire domain for each absorbing boundary edge[7], Y is the number of grid cells in the y direction, α_e controls the width of the envelope, and y_s controls the offset.[8] In general, the wider the Gaussian envelope is allowed to be, the less reflections of waves will be introduced back into the domain.

4.4 Program Pseudocode

In this section, pseudocode is given for an implementation of the hybrid scheme used to solve the chosen model on a static grid. This pseudocode is listed in Algorithms 4.4–4.7 and details a version using a third-order Runge-Kutta scheme and fifth-order WENO subscheme. All the necessary variables and constants needed in the program are assumed to be global for the sake of brevity.

Algorithm 4.4 is the main function of the hybrid program. It sets up all the initial conditions first, then adds a source on the first step and performs an entire Runge-Kutta time step before the source is zeroed out. This simulates an instantaneous source pulse. After the first time step has been calculated, the program loops until the number of desired time steps is reached.

[7]The integer n should be sufficiently large enough to fit the width Gaussian envelope, i.e. the envelope is wide enough for it to get acceptably close to 1.

[8]This offset is included because the index counters are included in the calculation and it gives the ability to offset the envelope correctly when using programming languages whose indices begin at either 0 or 1.

Algorithm 4.4: Pseudocode for the hybrid model main program.

```
1  HYBRID()
2  begin
3     initialConditions();
4     addSource();
5     rungeKutta();
6     H1 ← 0;
7     for step ← 2 to timeSteps do
8        rungeKutta();
9     end
10 end
```

The rungeKutta function in Algorithm 4.5 performs an entire time step. This consists of first calculating the right-hand side of Equation 3.1 and storing the result into the grid **K**. Then this **K** is used in the first stage of the Runge-Kutta. Once a stage of the Runge-Kutta is completed, the updateState function is called which simply updates values in the pressure and temperature grids. The boundary conditions must then be enacted. The boundaryConditions function should operate on $\mathbf{w}^{(n)}$. This is repeated for the other two stages of the third-order Runge-Kutta, and lastly, $\mathbf{w}^{(n)}$ is copied to **w** which is then considered the solution after the current time step.

Algorithm 4.5: Pseudocode for the hybrid model rungeKutta function.

```
1  rungeKutta()
2  begin
3     K ← rhs(); w^(n) ← w + Δt K;
4     updateState(); boundaryConditions();
5     K ← rhs(); w^(n) ← 3/4 w + 1/4 w^(n) + 1/4 Δt K;
6     updateState(); boundaryConditions();
7     K ← rhs(); w^(n) ← 1/3 w + 2/3 w^(n) + 2/3 Δt K;
8     updateState(); boundaryConditions();
9     w ← w^(n);
10 end
```

The rhs function in Algorithm 4.6 is where the DRP, WENO or hybrid schemes are used. Structuring the program in this manner allows this function to be replaced by other versions of the function quite easily. The function could be replaced, for example by a different hybrid scheme which implements the third-order WENO subscheme instead. In the pseudocode, **F**, **G** and **H** are all calculated first, and then

stored. Next the program loops through the entire grid and calculates the smoothness indicators for the hybrid scheme which are listed as Equation 3.34. If the smoothness indicator considers the cell to be not smooth (or nearby cells as well), then it is tagged for the WENO scheme. If the cell is considered smooth, then it is not tagged and the DRP scheme is used. When $r_x \geq 0.9999$ or $r_y \geq 0.9999$, then that location is considered to be smooth, and consequently the DRP scheme would be applied at that location, and when $r_x < 0.9999$ and $r_y < 0.9999$, the WENO scheme would be applied. The hybrid smoothness calculation can be put before the main loop in RHS and the tag values could be stored as their own grid as well. This would allow the program to tag neighboring cells to the cells that have been deemed not smooth, if desired to have a buffer zone around the not smooth cells, for example. Then the grid of tags could just be checked in the main loop in RHS. Another way would be to calculate the smoothness indicators of neighboring cells which can avoid storing cell tags. The hybrid scheme must be applied to each element of the vectors.

The WENO scheme is enacted through the use of two separate functions listed in Algorithm 4.7, which are WENOM and WENOP, that are used for the negative flux and positive flux, respectively, after the flux splitting. The flux splitting is performed at lines 10 and 14 of Algorithm 4.6.[9] It can bee seen that for each cell, the WENOM and WENOP functions are called a total of eight times to calculate the numerical flux at the left, right, top and bottom faces. Each function takes an array of values as input and each returns a single result after carrying out the WENO calculations. The program will spend most of its time in these two functions as they are called a significant number of times and both contain several floating point calculations. Therefore, it is important to focus on optimization of these functions.

Pseudocode for the INITIALCONDITIONS, ADDSOURCE and BOUNDARYCONDITIONS functions have not been given here as they are more straightforward, but versions of them can be seen in Appendix B. For a lightning channel source, the ADDSOURCE function would call the LIGHTNINGCHANNEL and DRAWSOURCE functions in Algorithm 4.1 and Algorithms 4.2–4.3, respectively. With this pseudocode, the reader should have an idea of how the entire model can be implemented in an efficient and structured manner in whatever programming language or tools that are desired.

While the pseudocode given in Algorithms 4.4–4.7 is focused on succinctness and clarity, an actual implementation programmed using MATLAB is given in Appendix B. The MATLAB implementation is structured in such a manner that it should be a straightforward process to port the code into a more performance-oriented language as well as having a focus on minimal memory usage. It chooses to recalculate certain values rather than store them (namely when calculating $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{H}$). Although recalculating these values typically results in a lot of division oper-

[9] In this implementation, a constant global flux splitting is used and therefore α does not need to be calculated in this case.

Algorithm 4.6: Pseudocode for the hybrid model RHS function.

```
1  RHS()
2  begin
3     calculate F, G and H;
4     for i ← 4 to X − 3 do
5        for j ← 4 to Y − 3 do
6           tag cell by using the hybrid smoothness indicator;
7           if tagged then
8              ii ← i − 1;
9              for q ← −2 to 2 do
10                fg_q = (1/2)(F_{ii+q} + α w^{(n)}_{ii+q,j});
11             end
12             pf_1 ← WENOP(fg); ii ← ii + 1;
13             for q ← −2 to 2 do
14                fg_q = (1/2)(F_{ii+q} − α w^{(n)}_{ii+q,j});
15             end
16             mf_1 ← WENOM(fg);
17             repeat lines 9–16, getting pf_2 and mf_2;
18             repeat lines 8–17 using G and in the y direction;
19             K_{i,j} = (1/Δx)(pf_1 + mf_1 − pf_2 − mf_2) + (1/Δy)(pg_1 + mg_1 − pg_2 − mg_2) + H_{i,j};
20          else
21             K_{i,j} = −(1/Δx) Σ_{m=−3}^{3} a_m F_{i+m,j} − (1/Δy) Σ_{m=−3}^{3} a_m G_{i,j+m} + H_{i,j};
22          end
23       end
24    end
25    return K;
26 end
```

ations, which are somewhat slow operations for the computer, this will be ideal in the AMR implementation to reduce the necessary amount of data that would need to be communicated to neighboring processes at each stage of the Runge-Kutta. The pseudocode is also structured in such a way that makes it simpler to port into an AMR version of the program as discussed in the next section.

4.5 Adaptive Mesh Refinement

When dealing with a large range of scales for both the domain and time such as in this work, efficiency of the execution of the model in the computer is crucial. In this model, large areas of the domain (mostly at the beginning of the simulation) will

Algorithm 4.7: Pseudocode for the hybrid model WENOM and WENOP functions.

WENOM(fg)
begin
  $\beta_0 \leftarrow \frac{13}{12}(fg_0 - 2 \cdot fg_1 + fg_2)^2 + \frac{1}{4}(3 \cdot fg_0 - 4 \cdot fg_1 + fg_2)^2$;
  $\beta_1 \leftarrow \frac{13}{12}(fg_{-1} - 2 \cdot fg_0 + fg_1)^2 + \frac{1}{4}(fg_{-1} - fg_1)^2$;
  $\beta_2 \leftarrow \frac{13}{12}(fg_{-2} - 2 \cdot fg_{-1} + fg_0)^2 + \frac{1}{4}(fg_{-2} - 4 \cdot fg_{-1} + 3 \cdot fg_0)^2$;
  $\zeta_0 \leftarrow \frac{1/10}{(\beta_0+\epsilon)^2}$; $\zeta_1 \leftarrow \frac{6/10}{(\beta_1+\epsilon)^2}$; $\zeta_2 \leftarrow \frac{3/10}{(\beta_2+\epsilon)^2}$;
  $\omega_0 \leftarrow \frac{\zeta_0}{\sum_{n=0}^{2}\zeta_n}$; $\omega_1 \leftarrow \frac{\zeta_1}{\sum_{n=0}^{2}\zeta_n}$; $\omega_2 \leftarrow \frac{\zeta_2}{\sum_{n=0}^{2}\zeta_n}$;
  $f_0 \leftarrow \frac{1}{6}(11 \cdot fg_0 - 7 \cdot fg_1 + 2 \cdot fg_2)$;
  $f_1 \leftarrow \frac{1}{6}(2 \cdot fg_{-1} + 5 \cdot fg_0 - fg_1)$;
  $f_2 \leftarrow \frac{1}{6}(-fg_{-2} + 5 \cdot fg_{-1} + 2 \cdot fg_0)$;
  return $\sum_{n=0}^{2}\omega_n f_n$;
end
WENOP(fg)
begin
  $\beta_0 \leftarrow \frac{13}{12}(fg_0 - 2 \cdot fg_1 + fg_2)^2 + \frac{1}{4}(3 \cdot fg_0 - 4 \cdot fg_1 + fg_2)^2$;
  $\beta_1 \leftarrow \frac{13}{12}(fg_{-1} - 2 \cdot fg_0 + fg_1)^2 + \frac{1}{4}(fg_{-1} - fg_1)^2$;
  $\beta_2 \leftarrow \frac{13}{12}(fg_{-2} - 2 \cdot fg_{-1} + fg_0)^2 + \frac{1}{4}(fg_{-2} - 4 \cdot fg_{-1} + 3 \cdot fg_0)^2$;
  $\zeta_0 \leftarrow \frac{3/10}{(\beta_0+\epsilon)^2}$; $\zeta_1 \leftarrow \frac{6/10}{(\beta_1+\epsilon)^2}$; $\zeta_2 \leftarrow \frac{1/10}{(\beta_2+\epsilon)^2}$;
  $\omega_0 \leftarrow \frac{\zeta_0}{\sum_{n=0}^{2}\zeta_n}$; $\omega_1 \leftarrow \frac{\zeta_1}{\sum_{n=0}^{2}\zeta_n}$; $\omega_2 \leftarrow \frac{\zeta_2}{\sum_{n=0}^{2}\zeta_n}$;
  $f_0 \leftarrow \frac{1}{6}(2 \cdot fg_0 + 5 \cdot fg_1 - fg_2)$;
  $f_1 \leftarrow \frac{1}{6}(-fg_{-1} + 5 \cdot fg_0 + 2 \cdot fg_1)$;
  $f_2 \leftarrow \frac{1}{6}(2 \cdot fg_{-2} - 7 \cdot fg_{-1} + 11 \cdot fg_0)$;
  return $\sum_{n=0}^{2}\omega_n f_n$;
end

contain ambient variable values, while the area of the domain where the lightning strike will virtually happen will contain a large amount of wave propagation activity which should be treated with more scrutiny, as they are of most concern in the simulation. In the static grid approach, the computer's computational effort will treat all of the areas of the domain with the same consideration, in regards to the grid, which is not an advantageous property. *Adaptive mesh refinement* (AMR) was specifically created to accommodate these large range of scales in finite difference models, and a successful implementation of AMR can decrease the computational cost of a model while also increasing the accuracy. The use of AMR results in the ability to focus computational effort and vary the solution accuracy on parts of the domain in which it will be most beneficial. In addition to the hybrid scheme, the technique of AMR is utilized to allow for simulation on a large domain. While the

idea of AMR is introduced here, its implementation is detailed further in Section 4.5.2.

The idea of refinement on a grid was used early on in other simulations with the realization that cells did not necessarily need to be homogeneous in size and that it is possible to have a specifically defined spatially-varying mesh discretization. An example would be to have the size of cells increase linearly in the x direction, which would result in higher resolution at the left side of the domain. However, in that case the mesh would be static, while in this model it is obvious that propagating waves will travel over time. It is clear that the mesh should *adapt* to the waves' behavior and locations. AMR was first realized through the work of Berger[51][52]. Since then, the idea has expanded for more than just finite difference schemes as Berger describes it for, e.g. finite volume methods on irregular meshes can benefit from AMR. The basic components necessary for an AMR-based algorithm are a base grid, a solver, an error or gradient indicator to tag cells for refinement or coarsening, some kind of grid manipulator, and a method for interpolation between disparate cells.

Mesh refinement can be accomplished by two different approaches: *structured* or *unstructured*. An unstructured approach means that the elements can vary in shape, size and orientation. Then the simulation would be handled using a finite volume or finite element method which typically calculates integrals over the elements. A structured approach means that the elements will be simple uniform shapes such as squares or rectangles. This allows for the use of finite difference methods as well, in which derivatives can be approximated, assuming the problem fits into the structured approach. A structured refinement approach is simpler than dealing with unstructured refinement, and usually requires less computation.

Structured refinement can also be be done in two ways: an *element-structured* fashion or a *block-structured* fashion. The difference between the two has to do with whether or not a group of cells tagged for refinement are refined individually (element-structured), or more cells around the tagged cells are tagged to form logical blocks of refined cells (block-structured). In the element-structured approach, a single grid level is used and the cells are typically stored using a quad-tree (2D) or oct-tree (3D) data structure. The numerical solver will need to be able to solve for each element on its own. An advantage to this approach is that it does not have to store overlapping neighborhoods of elements due to multiple grid levels. A disadvantage to this approach is that accessing the elements results in the traversal of a tree and they will not necessarily be stored contiguously in memory, causing an inefficiency.[10] Figure 4.9 gives a representation of the element-structured AMR approach. The block-structured approach involves multiple levels of grids in which the higher level grids are refined to use smaller cell sizes in parts of the domain deemed necessary. In the block-structured approach, the blocks of cells are known as *patches*. These patches

[10] Ji[53] attempts to counteract this through the use of a different data structure which reduces the amount of information necessary to store a tree structure called cell-based structured adaptive mesh refinement (CSAMR).

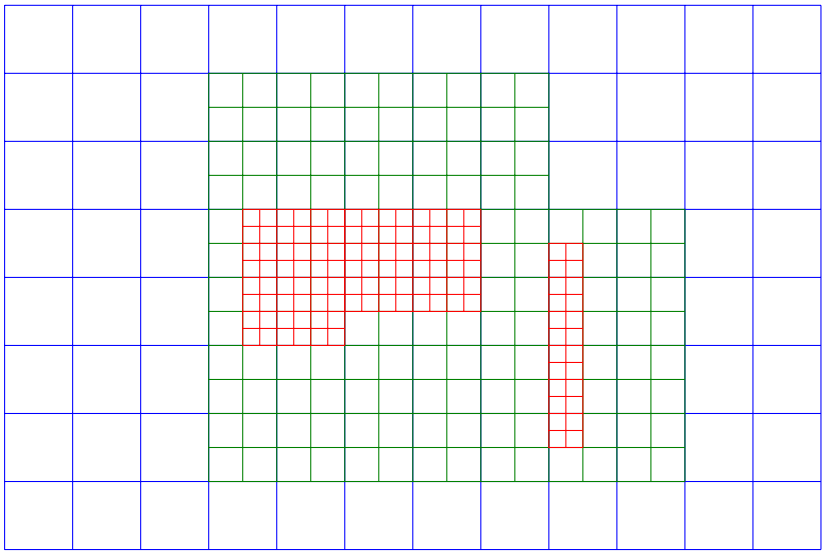

Figure 4.9: The element-structured AMR approach allows for a single non-overlapping grid that can be refined by storing the grid as a tree where subcells are children of their unrefined parent cell. When a cell is tagged for refinement in this case, the cell is divided into 4 subcells by using a factor of 2.

overlap the cells in the coarser grid that are to be refined and one requirement is that patches must be a subset of the grid on the next lower level. Figure 4.10 shows a non-conforming block-structured AMR grid hierarchy. These patches are then thought

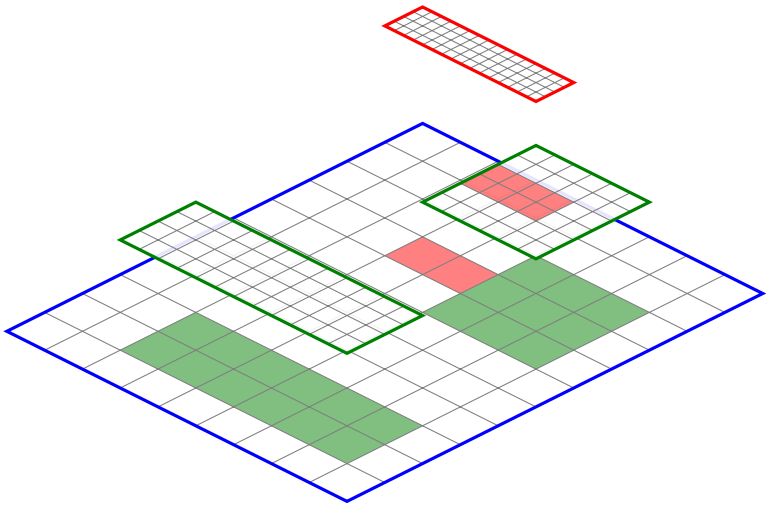

Figure 4.10: A figure displaying a non-conforming block-structured grid due to the red patch having cells existing outside of the patch on the next lower level. The higher level grids must be subsets of the lower level grids.

of as their own grids and should be designed to be integrated individually. When neighboring patches need data from each other, a synchronization between them occurs. When patches at levels of refinement above the base grid have no neighbors, then an interpolation at its boundaries occurs using data from coarser cells at lower levels to fill its ghost cells. Figure 4.11 shows the scenarios of how ghost cells are updated through synchronization, interpolation or the physical boundary conditions. During each time step, the solutions on the finer resolution grid levels take precedence

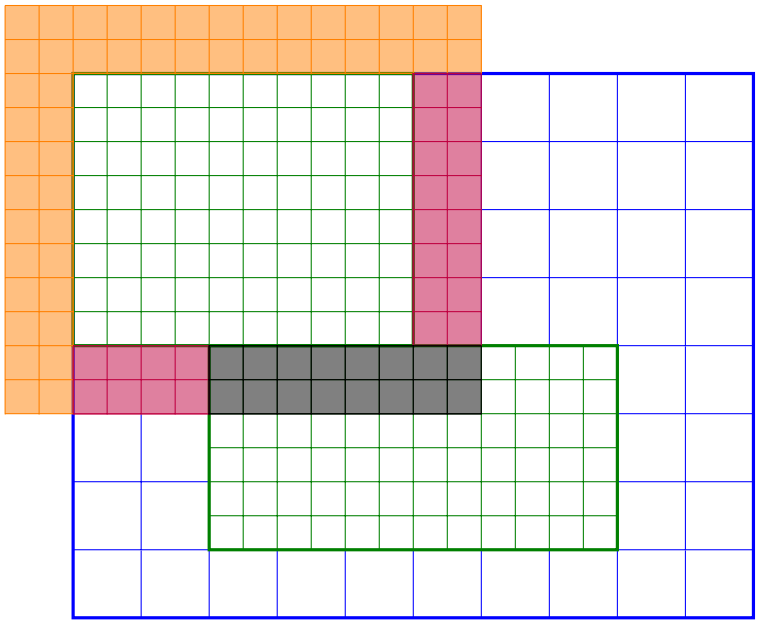

Figure 4.11: A display on how ghost cells for patches are updated at each time step. In this example, two ghost cells are used on each patch (in green and blue) and the ghost cells of only one patch (on one level of refinement) are shown. The cells in orange are taken from the physical boundary conditions. The cells in purple are interpolated from the cells on the lower grid level. Lastly, the cells in gray are synchronized between the neighboring patch.

over the solutions on the coarser grids and are generally copied down to lower level grids using some defined procedure. Figure 4.12 gives a representation of the block-structured AMR approach.

When using AMR, it is possible for refinement to happen in time as well as space. In Section 3.4 the CFL condition imposed on the time step, Δt, was discussed, which is written as

$$\Delta t = \text{CFL}\frac{\Delta x}{c}. \tag{4.8}$$

In this equation it can be seen that larger cell discretizations allow for larger time steps. This idea can be utilized in an AMR algorithm to decrease the amount of computations necessary and allow coarser grids to use larger time steps while finer

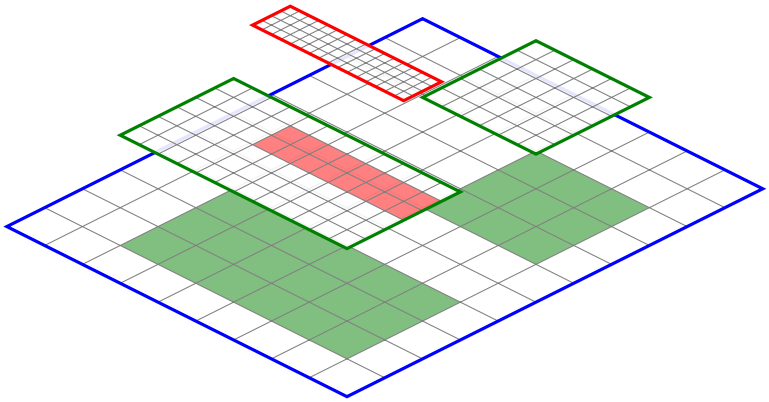

Figure 4.12: The block-structured AMR approach begins with a base grid and when cells are tagged for refinement, neighboring cells are tagged as well to produce logical blocks known as patches. These patches overlap the coarser grids.

grids use smaller time steps. This means that coarser grid levels do not need to be integrated as often as finer grid levels. This is known as using a *local* time step, while integrating each level using the same time step (according to the smallest stable time step at the finest grid level) is known as a using a *global* time step.

In this work, a block-structured AMR approach is utilized on the model. The block-structured approach was chosen due to its efficiency and scalability (especially toward a future GPU implementation as discussed in Chapter 6) as well as there being a more mature toolset available for this approach as opposed to the element-structured approach. Admittedly, an element-structured AMR approach may allow a wider range of scales to be managed than a block-structured approach. This is due to the issue that the source should not necessarily be allowed to be ill-defined on the base grid in the block-structured approach leading to the necessity for more elements to exist. Therefore a high efficiency may be possible by using an element-structured approach as well, but a block-structured approach will adapt to a high-performance computing environment in a superior fashion.[11]

The main modification to adding AMR into the program described in Section 4.4 will be the replacement of the RUNGEKUTTA function in Algorithm 4.5 with a function called STEP, listed in Algorithm 4.8. Although an AMR algorithm may not be ordered in the exact manner as Agorithm 4.8, depending on the circumstances, this STEP function describes how an AMR algorithm would generally be structured. The program with AMR included is now a time advancement of a *hierarchy* of grids, instead of just a single grid. The base grid will be designated as level 0, while the next level of refinement will be level 1, and so on. In the STEP function in Algorithm 4.8, the first step is to perform the Runge-Kutta on the current grid level, beginning at level 0. Next, if the current level is less than the number of possible

[11]This is discussed further in Section 4.5.2 and Chapter 6.

Algorithm 4.8: Pseudocode for advancing one time step in an AMR algorithm.

```
1  STEP(grids, level)
2  begin
3      RUNGEKUTTA(grids_level, level);
4      if level < maxLevels − 1 then
5          for n = 1 to rfactor do
6              STEP(grids, level + 1);
7          end
8          PROJECT(grids_level+1);
9          REGRID(grids_level+1);
10     end
11 end
```

refinement levels $maxLevels$, then STEP is recursively called on the next higher level a certain number of times, $rfactor$, which is the factor used when dividing a coarser cell into finer cells. This will integrate the grid hierarchy in the manner depicted in Figure 4.13. After those functions return, the function PROJECT is called on the

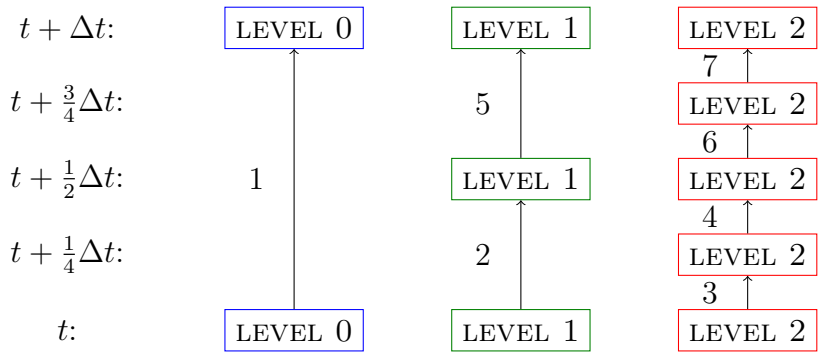

Figure 4.13: The single time step advancement of a grid hierarchy in AMR when refinement is done in time as well as space. The numbers next to each arrow are the order in which the substeps are taken.

next level up. Since the data on the higher level has been updated, this data then gets *projected* (this operation is sometimes referred to as *restriction* as well) down into the coarser level, modifying the values there with higher resolution data. This projection can happen in different ways, according to the manner in which the location of the stored solution values have been chosen, as well as the type of method used to integrate the grids. If the stored solution values are node-centered, then the solutions on the coarser grids line up directly with solution values on the higher level grids and the projection can happen by just copying the higher resolution solution values

directly onto the coarser grid. If the solution values are cell-centered, then the cell averages can be projected down to the coarser grid. In a finite difference scheme, a node-centered solution copying approach or cell average projection can result in a loss of mass between grid levels which is most prominent if the high amplitude waves are allowed to exit the finer level grids meant to capture them. It should be noted that it is also possible to project the fluxes into the coarser grids before the derivatives are calculated. This method could be added at an increased complexity. For a finite difference scheme, this generally means that the entire grid hierarchy can allow for mass conservation. Shen *et al.*[54] designed a method for distributing mass inconsistencies during the projection but found that their method would introduce non-physical solutions if high mass inconsistencies are encountered. Therefore they chose to tolerate the loss of mass conservation in the node-centered direct copy method and found errors to not be significant as long as shock waves were not allowed to exit the higher levels of refinement. This is discussed further in Chapter 6. Figure 4.14 shows a node-centered approach while Figure 4.15 shows a cell-centered approach of solution locations. When the projection is done, the REGRID function is called

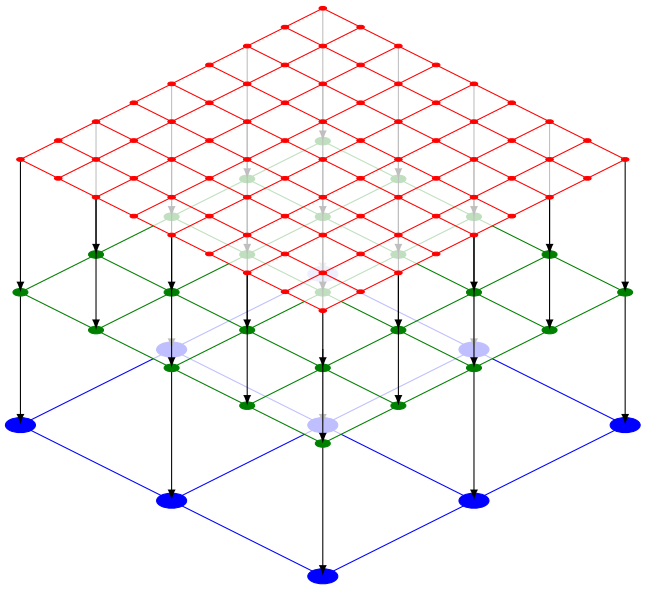

Figure 4.14: The storage of solutions at the nodes in an AMR grid hierarchy using a refinement factor of 2. During the projection operation, one option is to directly copy the solutions at higher levels to lower levels. This naive approach generally results in a loss of mass conservation.

which enacts the adaption process. The goal of the adaption process is to always keep discontinuities covered by higher resolution grids as they travel. The function REGRID accomplishes this task by calculating the gradient or error estimator over

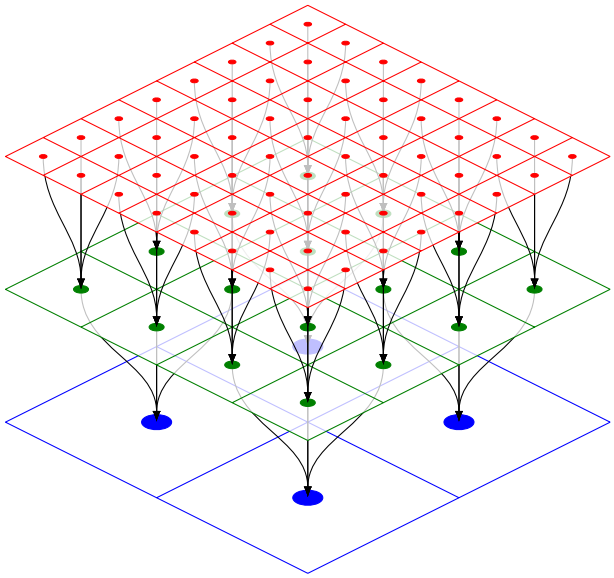

Figure 4.15: The storage of solutions at the cell centers in an AMR hierarchy using a refinement factor of 2. During the projection operation, the cell averages at the higher levels are copied to the lower level cells. Although this operation results in a possible loss of mass conservation, a more complex approach is to copy the fluxes at the cell walls, which would result in a conservation of mass.

each grid in order to tag cells which should be refined or not tagging cells which can then be coarsened. The adaption process also handles the clustering of tagged cells into rectangular patches.

There are further operations that will also need to occur within the PROJECT and REGRID functions as well as modifications within the RUNGEKUTTA function which would need to occur in a successful AMR algorithm that will be excluded from discussion for brevity. For example, the updating of ghost cells should happen at each stage of the Runge-Kutta.

For a more in-depth discussion of using WENO schemes with AMR, the reader is referred to Yoon[55], Manzanares[46], Shen *et al.*[54], and Li[56]. For other relevant discussions on AMR the reader is referred to Berger and Oliger[52], Berger and LeVeque[57], Berger and Colella[58], Jain[59], Srinivasa[60], and Chandra *et al.*[61].

4.5.1 AMR Libraries

Implementing AMR into the existing program directly would likely take a team of people to accomplish such a task. This can be alleviated by choosing an existing available programming library that abstracts the necessary AMR facilities from the programmer. There are several libraries available that would accommodate the expansion of the described program into the realm of an adaptive grid, as well as moving

it from a serial to parallel platform. A few notable packages are mentioned here.

4.5.1.1 Paramesh

Paramesh[62][63] was developed mainly by Peter MacNeice and Kevin Olson for NASA. Its framework allows for AMR on structured grids using finite difference methods for the solution of partial differential equations through a set of FORTRAN 90 subroutines. It uses an element-structured implementation that represents the grid in a quad-tree (or oct-tree) data structure. It was developed in C and FORTRAN 90 and built on top of MPI, making it very portable and allowing for highly scalable parallel execution and load balancing.

4.5.1.2 LibMesh

The libMesh[64] package was developed mainly at The University of Texas at Austin by Benjamin Kirk along with several other contributors. It is an AMR library that allows for the development of numerical simulations of partial differential equations on unstructured grids using finite element methods. It is developed in C++ and built on top of several existing and trusted packages such as PETSc and LASPack. It is capable of storing meshes in several standard formats and allows for parallel execution.

4.5.1.3 Clawpack

Clawpack was developed by several individuals. Most notably, Marsha Berger, a pioneer of AMR, and Randall J. LeVeque. Clawpack is a software package for solving hyperbolic conservation law problems using a set of FORTRAN routines. Within the package is another package called AMRClaw that handles the adaptive mesh refinement capability. AMRClaw is a block-structured AMR implementation. It is a very mature package, but does not have some of the important features of most of the other packages listed here, such as scalable parallel execution.

4.5.1.4 AMROC

AMROC was developed by Ralf Deiterding for solving hyperbolic partial differential equations using block-structured AMR. It is built on top of object-oriented C++ and MPI which allows it to scale to large parallel platforms. It is also built on top of Clawpack and the DAGH (Distributed Adaptive Grid Hierarchies) package developed by Manish Parashar and James Browne. AMROC allows the user to focus on the insertion of their own numerical methods while the package manages most of the other aspects of the grid hierarchy, as well as other aspects such as refinement in time.

4.5.1.5 BoxLib

Developed at Lawrence Berkeley Laboratory, BoxLib is an AMR framework that supports block-structured AMR, with optional refinement in time as well. It is written in a combination of C++ and FORTRAN and its parallel platform capability is based on MPI as well as OpenMP and has been shown to scale to hundreds of thousands of processors. It has support for parallel I/O, parallel checkpointing and restarting, as well as output for visualization to a standard file format, HDF5.

4.5.1.6 Chombo

Chombo is a library developed by Lawrence Berkley Laboratory for block-structured AMR on rectangular grids, implementing finite difference methods for the solution of partial differential equations. Chombo is built on top of an updated version of BoxLib called BoxTools which gives it a lot of the same capabilities and scalable infrastructure as BoxLib such as checkpointing and restarting simulations. It includes support for parallel execution and output to the HDF5 file format. Applications can be written using a mix of C++ and FORTRAN. C++ is typically used for the main program and FORTRAN is used for the *numerical kernels* that integrate each patch. FORTRAN was known to be easier for compilers to optimize. However, this is not so much the case as C++ compilers progress.

4.5.1.7 SAMRAI

SAMRAI was developed by Lawrence Livermore National Laboratory and stands for Structured Adaptive Mesh Refinement Application Infrastructure. SAMRAI is very similar to Chombo in its infrastructure and quality. It is designed through object-oriented programming and C++, along with an interface to FORTRAN for the numerical kernels. It supports output to standard HDF5 files for visualization, support for reading structured input files and support for checkpointing and restarting simulations. Although many AMR packages have ceased development some years ago relative to the time of this work, SAMRAI was updated most recently (2010). Version 3.1.0-beta of SAMRAI is currently the most modern block-structured AMR package and is used for the AMR implementation in this work.

4.5.2 Implementing AMR using SAMRAI

This section is used to detail the process of implementing block-structured AMR into the model using the SAMRAI framework. The SAMRAI framework is a fully object-oriented collection of C++ classes, and therefore it will be assumed that the reader is relatively familiar with the C++ object-oriented programming paradigm. SAMRAI programs are typically comprised of a main program that sets up and handles the utilities of the program outside of the numerical scheme, a user-created class that

contains the model and methods for solving, and finally an optional interface to FORTRAN as well as the FORTRAN procedures for the numerical kernels that are used to integrate each patch. The user-defined class uses virtual functions that the user can explicitly define to accommodate their chosen numerical methods. SAMRAI then calls these user functions when needed, in a certain order. There is a large amount of detail and auxiliary code necessary for a fully-functioning program using SAMRAI. Much of this code, used for writing output files, reading input files, checkpointing, or restarting from a checkpoint for example, is omitted as it is not necessary for understanding the basic implementation of the model. Since the program using SAMRAI must be written in C++ and optionally, FORTRAN, indices will begin at 0, where previously, MATLAB was being referenced whose indices begin at 1.

When implementing AMR into the model, the situation arises of how the hybrid model should be utilized. AMR must employ its own smoothness indicator to decide which cells should be tagged for refinement. Consequently, it knows which areas of the domain are smooth and which are not. This is the same idea as the hybrid scheme. With this being the case, it is possible to use the hybrid smoothness indicator as the AMR smoothness indicator as well. At each level in the grid hierarchy, there will be areas of the domain that are considered either smooth or not smooth for that level. However, according to the base grid, any cells tagged for refinement are considered not smooth and therefore the WENO scheme will be imposed on every level above the base grid. The hybrid scheme will be used on the base grid as it covers the whole domain, and the WENO scheme should still be used over areas that are not smooth. Even though the solutions at higher levels will eclipse solutions at lower levels as a result of projection, unphysical solutions should still be avoided at lower levels, meaning that the initial conditions on the base grid should be well defined. Figure 4.16 gives a graphical representation of the model now using an AMR grid hierarchy.

Before development, a decision concerning use of SAMRAI's capabilities regarding the chosen methods is made; it is in regards to SAMRAI's built-in algorithms for scheduling calls to certain functions to correctly march the simulation through time. SAMRAI has two main integrator algorithms to handle this. One is the *hyperbolic integrator* and the other is the *method of lines integrator*. The hyperbolic integrator will enable refinement in time, but does not specifically allow the use of a Runge-Kutta method. The method of lines integrator (MOL) is used for employing a Runge-Kutta scheme in time, and therefore this algorithm is chosen. A slight disadvantage to the MOL integrator is that it uses a global time step. This will cause an increase in runtime execution of the program, but will make recording parts of the domain using virtual microphones simpler in that the time step can be fixed to the sample rate of a wave file, such as 11025 or 22050 Hz for example, and no dithering or interpolation will be necessary for the recorded data. Pseudocode of the MAIN function of the AMR program is given in Algorithm 4.9.

The basis of the MAIN function in Algorithm 4.9 is to set up SAMRAI in the desired

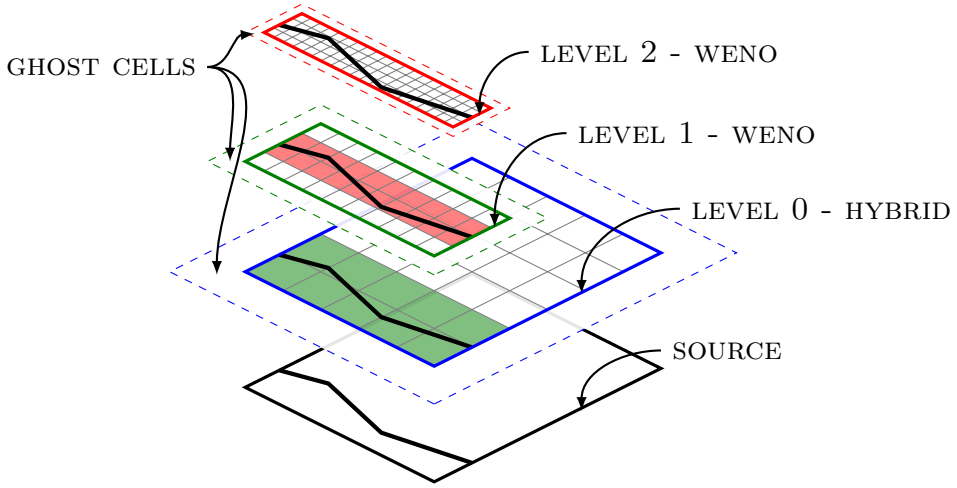

Figure 4.16: A graphical depiction of three levels of AMR applied to a simple lightning channel geometry at the initial condition. The dashed borders around each level denote the location of the ghost cells on that level. The hybrid scheme is implemented on the lowest level while the WENO scheme is applied to all higher levels.

Algorithm 4.9: Pseudocode for the MAIN function of the AMR model.

```
1  MAIN()
2  begin
3      Set up SAMRAI;
4      Create a model object instance;
5      Register MOL integrator;
6      for step ← 0 to timeSteps do
7          if shouldAddSource then
8              while stillRefine do
9                  ADDSOURCE();
10             end
11         end
12         ADVANCEHIERARCHY();
13         if addedSource then
14             SETSOURCETOZERO();
15         end
16         if shouldRegrid then
17             REGRID();
18         end
19     end
20     Shut down SAMRAI;
21 end
```

fashion, create an instance of the model object, register the MOL integrator and advance the grid hierarchy in a loop until the desired amount of time steps is reached. In this main loop, the source will be added at the desired time steps with the extra work of checking whether the grid should be refined at any places and then adding the source using the lightning channel algorithm on any newly created patches in higher refinement levels. This is repeated until the highest level of refinement is reached. Next, the loop calls ADVANCEHIERARCHY, which advances the grid hierarchy one time step forward. If a source was added then the source grid is set to all zeros. Once that is done, a condition is checked to determine whether REGRID should be called which would then run the adaption process to reconfigure the grid hierarchy according to the updated values.

When setting up SAMRAI, the variables that need to be stored on a grid are registered with SAMRAI's variable database by the function REGISTERMODELVARIABLES in the model object's public member functions. In this function, a grid for $\mathbf{w}^{(n)}$[12], pressure, temperature, source, as well as a grid for storing the temporary values of the right-hand side of the main equation are registered. For the fifth-order WENO scheme, three ghost cells are necessary around the border, while for the third-order WENO scheme, two ghost cells are necessary. The fourth-order DRP scheme needs three ghost cells as well, therefore the grids are all registered with three ghost cells. When registering the grid variables, the MOL integrator must know whether the variables are of type SOLN or RHS. Variables of type SOLN are stored at all times while variables of type RHS are stored temporarily during the Runge-Kutta and freed after use. The name for the grid that will be registered as type RHS will be **K**. All other grid variables will be registered of type SOLN. These grid variables must also be given a refinement strategy and coarsening strategy. These refinement and coarsening strategies are associated with where the solutions on the grid are stored, e.g. node or cell-centered. The solution values are chosen to be cell-centered and therefore SAMRAI's cell-centered strategies must be used. The **K** grid will neither need to be coarsened nor refined and therefore will be registered with the "NO_COARSEN" and "NO_REFINE" strategies. All other grid variables should allow for refinement and coarsening by registration with the "LINEAR_REFINE" and "CONSERVATIVE_COARSEN" strategies, which will refine data with a linear interpolation and attempt to coarsen conservatively which, in this case, results in the averages of refined cells being copied to the coarser cells, respectively.[13]

Although the main function of the program controls the major program execu-

[12]This will be five or six grids in depth depending on the truncated or full model, respectively, and SAMRAI creates **w** as necessary before the Runge-Kutta so there is no need to register a **w** specifically.

[13]A node-centered approach should theoretically perform faster, however, this approach was tried and performed significantly slower than the cell-centered approach. It also became apparent that SAMRAI did not allow values on the grid of tags to be node-centered which may have had an effect on performance.

tion flow, the model object instance is where the majority of the code to implement the methods of the model is written. Several functions of specific names can be defined in the public section of the object that will be overloaded so that SAMRAI will call them in specific orders and execute the code given by the user. The major list of object member functions used to implement the model that SAMRAI will call in a specific order are: INITIALIZEDATAONPATCH, COMPUTESTABLEDTONPATCH, SINGLESTEP, TAGGRADIENTDETECTORCELLS, SETPHYSICALBOUNDARYCONDITIONS, and the object constructor.

Algorithm 4.10: Pseudocode for the INITIALIZEDATAONPATCH function of the SAMRAI program.

```
1  INITIALIZEDATAONPATCH()
2  begin
3     H(0) ← 0;
4     if initialTime then
5        p ← p_0; T ← T_0;
6        w(0) ← ρ_0; w(1) ← 0;
7        w(2) ← 0; w(3) ← 0;
8        w(4) ← T_0 · ρ_0;
9     end
10 end
```

Both the model object constructor and the INITIALIZEDATAONPATCH function will establish the initial conditions with the model constructor initializing the single variables while INITIALIZEDATAONPATCH initializes the variables on the grid hierarchy. SAMRAI will invoke the INITIALIZEDATAONPATCH function any time a new patch is added to the hierarchy. This function is given in Algorithm 4.10. In the function, the source grid ($\mathbf{H}(0)$) should always be initialized to zero. If this is not done, when the source is added and then the refinement process run, if refinement is to occur, the source is drawn on the new level and if the newly inserted patches are not initialized to zero, SAMRAI will instead interpolate the data from lower levels. This can cause a smearing in the source grid as shown in Figure 4.17. The rest of the grids are set to their initial conditions only at the first time step in the simulation.

The COMPUTESTABLEDTONPATCH function will calculate a Δt to return for each patch through a CFL condition, or it can return a global time step. In the simulations recording thunder for audio playback, it will return $1/wavSampleRate$ where $wavSampleRate$ is the frequency of the audio wave file to be generated. If using a standard of 11025, 22050, or 44100 Hz, etc. then the data can be offset from the ambient pressure and written directly to a wave file without dithering or interpolation.

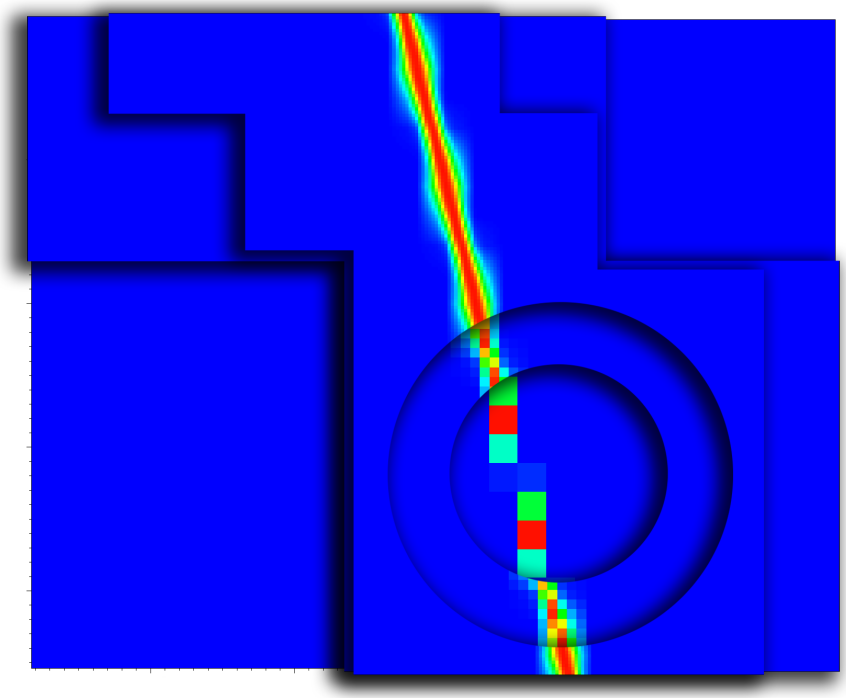

Figure 4.17: A three-level grid, cut away to see the segment on each level. Smearing of the source segments can occur if new source grid patches are not initialized to zero properly. An exaggerated example is shown with the source being ill-defined on the base level.

The SINGLESTEP function is designed to take one step of the Runge-Kutta, meaning each call to SINGLESTEP will accomplish a single stage of the Runge-Kutta. This allows the specific Runge-Kutta method that will be employed to be designated in the input file for the program. A second-order or third-order Runge-Kutta can then be specified without recompiling the program. The SINGLESTEP function is structured as shown in Algorithm 4.11. This function will be called for each patch.

In Algorithm 4.11, four other functions are called. The RHS function is written in FORTRAN and is structured similar to its previous version listed in Algorithms 4.6–4.7. The RUNGEKUTTA function updates the $\mathbf{w}^{(n)}$ solution vector using the updated values on the right-hand side of the main equation which are given in $\mathbf{K}$. This function is listed in Algorithm 4.12 where α_1, α_2, and β are values SAMRAI passes into the function that are given in the input file. An example of the MOL section of the input file which would define the use of the second-order Runge-Kutta scheme is given in Figure 4.18. Lastly, the function UPDATESTATE merely updates the pressure and temperature grids using Equations 2.8 and 2.9, respectively.

The function SETABSORBINGBOUNDARYCONDITIONS is a user-created function that SAMRAI does not know to call. It is invoked at the end of each Runge-Kutta

Algorithm 4.11: Pseudocode for the singleStep function of the SAMRAI program.

```
1 singleStep()
2 begin
3    Call rhs FORTRAN procedure;
4    Call rungeKutta FORTRAN procedure;
5    Call updateState FORTRAN procedure;
6    Call setAbsorbingBoundaryConditions procedure;
7 end
```

Algorithm 4.12: Pseudocode for the rungeKutta function of the SAMRAI program.

```
1 singleStep()
2 begin
3    w^(n) ← α1 w + α2 w^(n) + β Δt K;
4 end
```

```
MethodOfLinesIntegrator{
   order       = 2
   alpha_1     = 1 , 0.5
   alpha_2     = 0 , 0.5
   beta        = 1 , 0.5
}
```

Figure 4.18: An example of the input file section for the MOL integrator defining the use of the second-order Runge-Kutta scheme.

stage to enact the absorbing boundary conditions on the current patch. It is necessary to call this function here because the other function that deals with boundary conditions, setPhysicalBoundaryConditions, is only called by SAMRAI on patches that touch the physical boundary of the domain. Since absorbing boundary conditions must happen on patches near the boundary, setAbsorbingBoundaryConditions must be called on every patch. The setAbsorbingBoundaryConditions function only handles the absorbing boundary condition operations, while setPhysicalBoundaryConditions handles the reflecting boundary conditions which only need to occur on patches that do touch the physical domain boundary. The boundary conditions are applied in the same manner as a static grid approach with care taken to calculate where the patches are located in the physical domain for the absorbing

boundary conditions.

The function tagGradientDetectorCells simply calls its FORTRAN procedure to tag cells on the patch for refinement. It checks the smoothness of the source grid ($\mathbf{H}(0)$) as well as the density grid ($\mathbf{w}^{(n)}(0)$) so that refinement will happen according to any source that has been inserted as well as any evolution of the system.[14] Separate tolerance values for the source grid and density grid are used since they will have fundamentally different amplitudes. To be clear, although the hybrid smoothness indicator is used in the tagging calculation for AMR, it is separate from the calculation that is independently done for the hybrid scheme at level 0 in the RHS function. Therefore, a different tolerance is used for the actual hybrid scheme on the base level as well.

Structuring the program in the way shown in Section 4.4 and Appendix B as well as adhering to the major guidelines illustrated in this section allow for a successful implementation of the model using AMR. This implementation allows for a minimal amount of storage necessary to accomplish the simulation (two solution vectors, a pressure grid, a temperature grid, a source grid, a grid of tags, and the grid for the right-hand side of the main equation) which is desirable as the minimal amount of ghost cell information will need to be communicated at each stage of the Runge-Kutta to other processes in the computer. Communication is typically a bottleneck when running large-scale parallel programs due to I/O being very slow relative to computation.

4.5.3 Load Balancing

An issue that occurs when distributing work over several concurrently running processes is the manner in which workload is distributed across the processes or processors in the machine. The patches in the case of AMR are thought of as *virtual* processes in that their calculations can occur independently until communication of ghost cells are necessary. However, these virtual processes are then distributed among the actual processes of the machine in a certain fashion known as *load balancing*. The virtual processes can create a granularity that allows the load balancer to distribute the computational workload evenly across processes (which the operating system's scheduler should then evenly distribute across the machine's processors). If load balancing were not to occur, then it is likely that certain processes would finish their calculations at each Runge-Kutta stage before other processes and then be obligated to wait for the communication step. It is possible that the workload among cells is considered *uniform* in that each cell is doing the same general calculations as any other cell in the system. Then the load balancer would create the patches and dis-

[14] Although calculating error estimators is the most reliable for any general problem, calculating error estimators generally requires more computations than calculating some kind of gradient. For this work, the smoothness indicator given in Section 3.3.3 will suffice for both the hybrid scheme at the base level as well as deciding which cells to refine at any level.

tribute them across processes in such a manner that each process contained roughly the same amount of cells on which to do the calculations. In the case of the AMR scheme described here, the workload is considered *non-uniform* in that certain cells may only do calculations for the DRP scheme, while others would need to spend more time to do calculations for the WENO scheme. If a uniform load balancing strategy were to be used, then processes that did not need to do many WENO scheme calculations would finish their calculations at each Runge-Kutta stage before the processes that had to do a fair amount of WENO calculations and then have to wait for the communication to occur. Therefore, a non-uniform load balancing strategy should be used to consolidate better efficiency. SAMRAI is equipped with the capability of both load balancing strategies. A tree-based strategy is used for the uniform load balancer, and a chop-and-pack strategy is used in the non-uniform load balancer and they are labeled as such. The chop-and-pack non-uniform load balancer can also pack neighboring patches into processes by calculating abstract spatial coordinates of them rather than packing by a more naive greedy algorithm. SAMRAI also has the ability to specify the manner in which it should chop up the domain. In this case, chopping the domain mainly across the y dimension would be of benefit, as the computational workload varies more across the x dimension than the y dimension. This can be done by forcing patch sizes to be long rectangles in the x dimension, or by specifying to the load balancer in the input file how to partition the domain across processes. Load balancing is another aspect of the model that can be tuned using several of the optional parameters that exist for it. Generally, there is a tradeoff between the overhead of the amount of communication necessary between patches at each step and the granularity at which the load balancer is able to pack the virtual processes into the actual processes when optimizing performance. However, an in-depth discussion of this optimization is not conducted. The reader is referred to Morris[65] and Wissink[66] for further details on non-uniform load balancing.

4.6 Summary

This chapter has described the necessary details of the model implementation for execution in the computer. A statistically based algorithm for creating a convincing random lightning channel geometry was given that is used to store the channel geometry as a list of segments. This list of segments is then given to another algorithm to draw the channel on the source grid through the use of an amalgamation of primitive mathematical functions which can draw the source in a sufficiently smooth manner with variable channel width, and at arbitrarily high resolutions for use in an AMR implementation of the model. This specialized algorithm avoids the use of using a large rasterized image file as a source, making setting the initial conditions a manageable task. The last area of discussion regarding the model was the boundary conditions. Both absorbing and reflecting boundary conditions were discussed as the domain

should have a reflecting lower boundary for the ground and absorbing boundaries on the left, right and top boundaries to simulate the waves' ability to exit the domain in those directions. Pseudocode was then given for a minimal version of the entire hybrid model that focuses on a specific structure for the program to be easily ported to an AMR program architecture. Lastly, adaptive mesh refinement was illustrated as it is crucial for the extension of the model into a larger domain. Block-structured AMR was chosen for its extensibility and a more straightforward porting from a static structured grid using a mature toolset. Although several frameworks for AMR exist, with most of them focused on block-structured AMR, the SAMRAI framework was chosen for it being the most modern as well as its performance capability, breadth of features and object-oriented approach. The most important details for adapting the model into SAMRAI's infrastructure were then specified, giving the reader the ability to scale the model into a high-performance parallel platform. The model and methods described are used to present the results of this work in the next chapter.

Chapter 5

Results

5.1 Introduction

As mentioned earlier, for a verification of the model, the reader is referred to Wochner[8]. The purpose of this chapter is to verify and investigate the newly added effects and methods introduced in this work and to garner results from utilizing the model for the generation of synthetic thunder. The effect of refraction is validated, along with the ability to introduce wind effects into the model. The hybrid scheme is verified as well as the use of AMR with the model. Comparisons between the third-order and fifth-order WENO schemes, and the second-order and third-order Runge-Kutta schemes used for solving the model are carried out. The performance of these schemes, along with the performance of the hybrid scheme and AMR are examined as well. Although the model and methods have been chosen for their high accuracy capabilities and range of accuracy/performance capabilities, performance and efficiency are held to foremost importance, above accuracy, specifically in this work. Lastly, a phenomenological study of the model's ability to effectively simulate thunder is given.

In this section, a shorthand notation for the different methods (and combinations thereof) discussed for solving the model is adopted. This shorthand has three main parts which are separated by dashes. The first will reference the order of the Runge-Kutta used in the method as RK3 for the third-order version and RK2 for the second-order version. The second part will reference the spatial derivate calculation method. The spatial methods are the DRP, third-order WENO, fifth-order WENO, hybrid, and AMR schemes. The hybrid scheme will use a fifth-order WENO subscheme and the AMR scheme will use the hybrid scheme at the first level and the fifth-order WENO scheme at all higher levels. A reference to an AMR scheme will have an additional number to denote how many levels of refinement are used. Although the AMR scheme can use a refinement factor higher than two for each level, at a possible increase or decrease in efficiency, a refinement factor of two will be used to avoid a very large range of scale which could put the stability of the model at execution at

risk. Lastly, a reference to whether or not global flux splitting is used in the WENO scheme is denoted by a 'G' and a number for the value of α, otherwise a local Lax-Friedrichs flux splitting is assumed. For example, RK3-AMR2, is a scheme using AMR with two levels of refinement and a third-order Runge-Kutta. RK2-WENO5-G360 denotes a fifth-order WENO scheme using a global flux splitting with $\alpha = 360$ and a second-order Runge-Kutta.

For comparison of schemes in regards to accuracy and performance, the example problems referenced in this chapter were run on an Apple iMac with an Intel i7 2.8 GHz quad-core processor with 8 GB of RAM. For performance analysis, all cases were run in a single process in order to avoid auxiliary factors such as load balancing issues and inter-process communication, etc. This particular processor employs a dynamic clock speed, meaning that the clock speed can be increased for a particular core if other cores are idle, so the performance analysis in this chapter should be heeded only in a general sense. The large simulations of thunder events referenced in this chapter were run on a supercomputer at The University of Texas at Austin consisting of 1888 compute nodes, each of which contained 2 hex-core Intel Xeon 5680 processors, along with 48 TB of aggregate memory.

5.2 Refraction

When simulating atmospheric sound propagation in a large-sized domain, one meteorological effect that plays a role is *refraction*. Refraction occurs as a result of the temperature gradient in the atmosphere as altitude increases. During daytime conditions, the air temperature usually decreases with altitude, which then decreases the speed of sound at higher altitudes. The result of this is that sound waves tend to bend upward as they propagate. As the sound waves bend upward, they can create a sound "shadow" a large distance away from the sound source. This effect is depicted in Figure 5.1. The sound shadow shown in Figure 5.1 would typically occur very far away from the lightning strike with a radius of curvature of about 55 km, but it depends on many factors[16]. An opposite effect sometimes occurs in a nighttime situation in which the temperature actually increases with altitude, which then increases the speed of sound with altitude. The result of this is the sound waves bending downward and reflecting off of the ground, and subsequently being bent downward again and etc. This increases the pressure and amplifies the effects that the ground plays in the system.

The effect of refraction is implemented in the model through a parameter called the *lapse rate* of the speed of sound in the atmosphere. Equation 2.8 is exploited by using a linear function of height within the speed of sound constant, c, rather than just a purely constant value, c_0. This is not physically realistic in a sense that other variables such as temperature, density, etc. are not varied with height. To do so would require the model to incorporate gravity in order to force the varied quantities

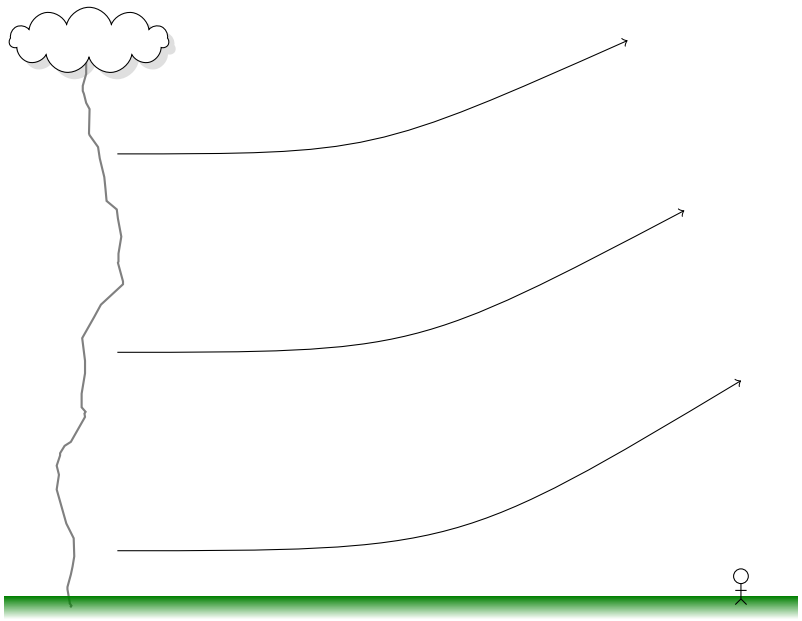

Figure 5.1: The typical daytime effect of refraction. Temperature decreases as altitude increases. Therefore the speed of sound is slower at higher altitudes. This causes the sound waves to bend upward as they propagate, effectively creating a sound "shadow" where the observer is located in this figure.

downward; otherwise the unequal pressure would propagate all quantities upward. This is discussed further in Chapter 6.

Temperature in the troposphere (the first layer of the atmosphere, which extends from the ground to 10–12km) ordinarily decreases by about 6.5 degrees Kelvin per kilometer, known as the lapse rate of temperature[67]. With ambient temperature at the Earth's surface ∼293 K, the speed of sound (c_0) at the Earth's surface is 343 m/s. The speed of sound in air can be calculated from temperature as:

$$c_{\text{air}} = \sqrt{\gamma RT}. \tag{5.1}$$

Traveling to 10 km from the Earth's surface, the ambient air temperature is then ∼230 K, making the speed of sound ∼304 m/s at that altitude. This equates to a decrease in the speed of sound at a lapse rate of $\sim -3.9 \times 10^{-3}$ m/s per meter.

To verify the effect of refraction in the model, a domain of 32 m × 10 m is used with a low amplitude, 100 Pa, vertical line source placed at 2 m from the left edge boundary. The effects of modified classical absorption and molecular relaxation are removed. Absorbing boundaries are used at the left and right edges, with reflecting boundaries at the top and bottom edges. An exaggerated lapse rate of -2 is used and the simulation is ran until the wave propagates to ∼30 m. Using a lapse rate of -2, the pulse at the lower edge of the domain should be traveling at 343 m/s while the

pulse at the upper edge of the domain should be traveling at 323 m/s. At 8.16×10^{-2} seconds, the lower end of the pulse should be at ∼30 m, while the upper end of the pulse should theoretically be at ∼26.4 m. Figure 5.2 shows the initial source and the pressure profiles at 0.08 s for a lapse rate of −2 and −4. It is apparent that the pulse at the upper edge of the domain after 0.08 s is located near 28 m, which is about half the amount expected for the effect. Other factors may need to be addressed as to why there is a lower than expected rate of refraction in the model. However, it is evident that exploitation of the c term in Equation 2.8 can be used to force an approximation of the effect of refraction into the model. To achieve the desired rate of refraction, a lapse rate for the speed of sound should be about double what the physically-based value is to be. For this reason, the lapse rate is chosen to be -6×10^{-3} as listed in Chapter 2. It is likely that lapse rates of other ambient values of variables such as temperature and density, etc. that are not included in this model have an effect on the lowered expected rate of refraction.

5.3 Wind

Wind is also another effect, although minor, that can have an impact on the thunder signature of a lightning strike. Wind is known to advect sound waves in the direction it is blowing. Wind can have very complex movement and is an area of research and simulation all its own. This section serves only a purpose of verifying the capability of the model to include wind. Therefore, a simplifying assumption is taken to verify its capability in the model in which the wind is blowing in one particular direction. Any more complicated situations are left to possible future work.

To incorporate the effect of wind, an extra force is added into the momentum equation (Equation 2.15). The momentum equation with a wind force term, $\mathbf{f}_w$, added is written as such:

$$\rho \frac{D\mathbf{v}}{Dt} = \mathbf{f}_w - \nabla p + \nabla(\mu_B \nabla \cdot \mathbf{v}) + \mu \sum_{ij} \mathbf{e}_i \frac{\partial \phi_{ij}}{\partial x_j}. \tag{5.2}$$

In particular, the model will be run with a force strictly in the x direction using the split momentum equation, with the equation corresponding to the x component. These types of forces are sometimes called *body* forces, and gravity, for example, is typically implemented into a Navier-Stokes model in this manner. A Gaussian-distributed wind force is introduced at the center of the domain in the y dimension as shown in Figure 5.3. The example problem used to verify the effect of refraction is run without refraction in order to observe the effect of the wind force. Since only a small time is simulated in the example problem, a highly exaggerated wind force traveling at ∼1000 m/s (a realistic wind velocity would be more on the order of 30 mph or 13.4 m/s) is used so that its effect is made more apparent. Figure 5.4 shows the pressure profile of the domain after 3.84×10^{-2} s of simulation. It is apparent that

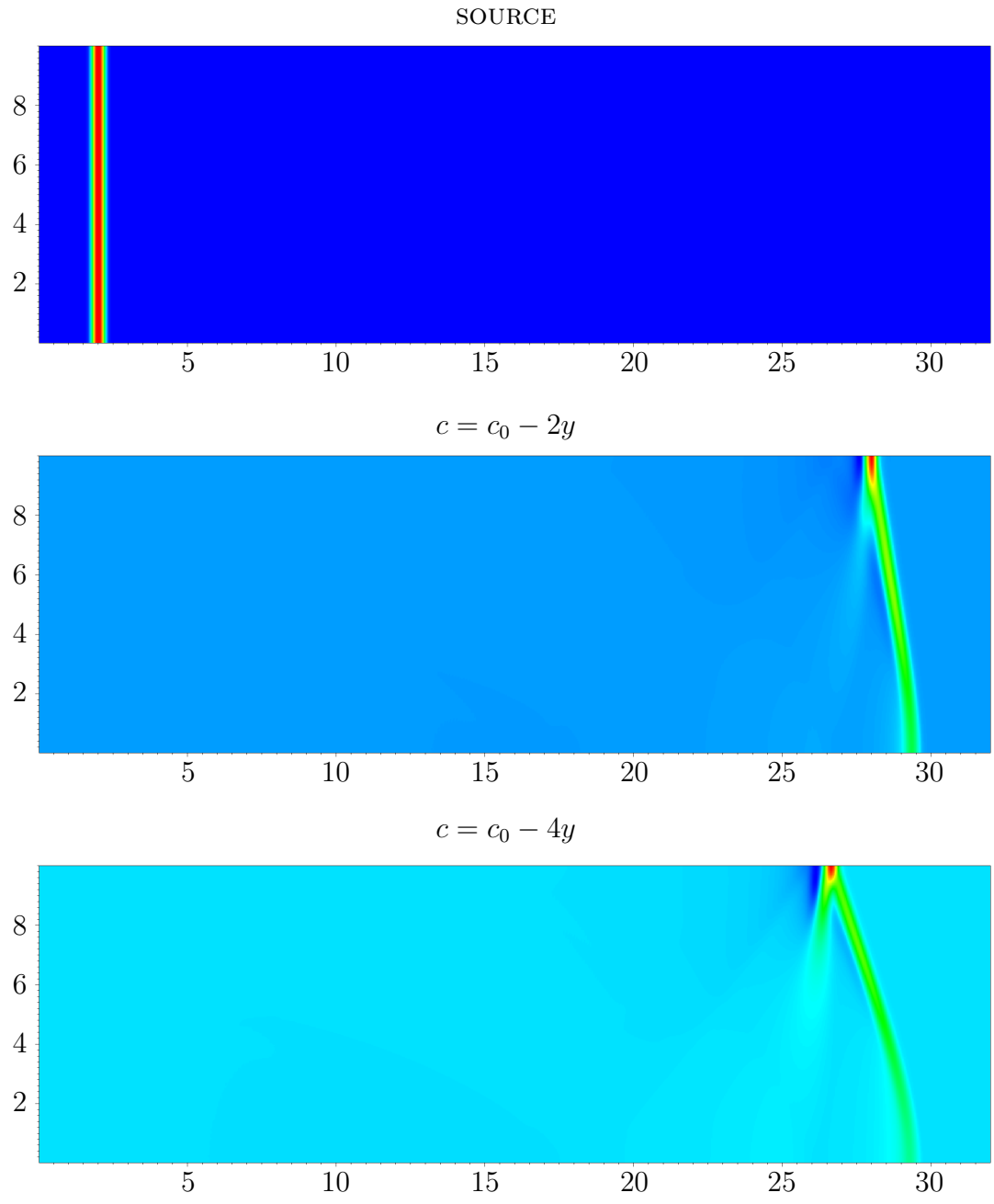

Figure 5.2: Pressure plots of the initial source and results of refraction with lapse rates for c of -2 and -4. The lower two plots are taken at a time of 0.08 s. The lower end of the pulse is at ∼30 m. The model puts the upper end of the pulse at ∼28 m for a lapse rate of -2, when it should be at ∼26 m as in the bottom plot.

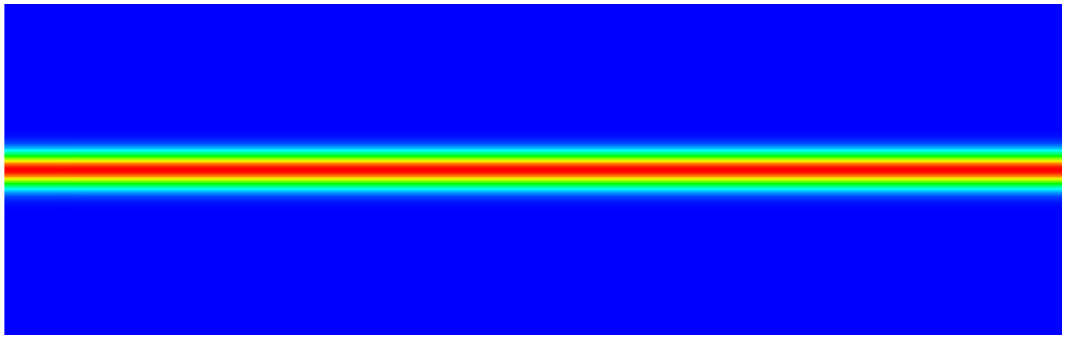

Figure 5.3: A wind source in the manner of a jet stream is used in the momentum equation associated with the x component to verify the model's ability to incorporate the effect of wind. An exaggerated wind force traveling at ∼1000 m/s is used to observe the effect in a short time frame.

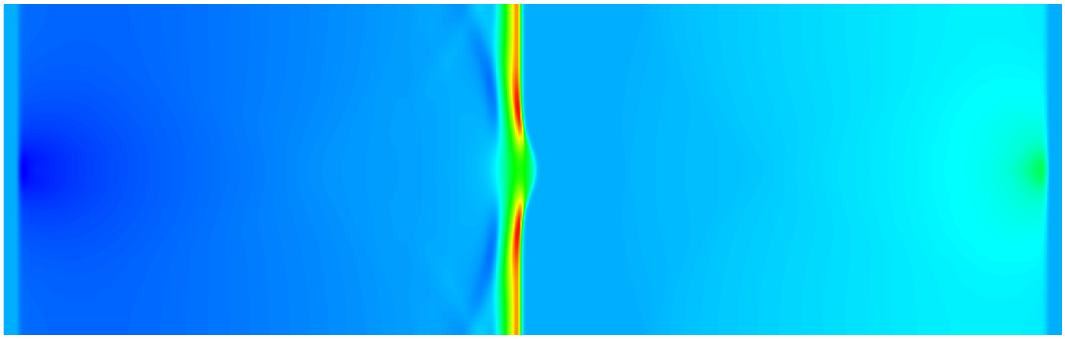

Figure 5.4: The pressure profile of the example problem at 3.84×10^{-2} s, after 4800 time steps. As expected, the center of the y dimension has experienced a higher rate of advection according to the wind source.

the middle of the segment has experienced a higher rate of advection as expected. Accordingly, wind forces in the y direction are made possible by using the momentum equation associated with the y component, and would allow for complex wind force configurations in any direction.

5.4 Verification of Hybrid Model

Before employing the model in the generation of thunder, the hybrid scheme is verified. Because analytical solutions to any example problem do not exist for this model, the hybrid scheme is verified against the RK3-WENO5 scheme on a static grid and it will be assumed that this scheme produces a sufficiently accurate solution to the problem for verification. A different simple example problem than the one previously used in this chapter is constructed for verification and comparison purposes for the various

schemes introduced for solving the model. The initial condition for the constructed problem can be seen in Figure 5.5. A domain of size 32 m × 20 m is used with a grid

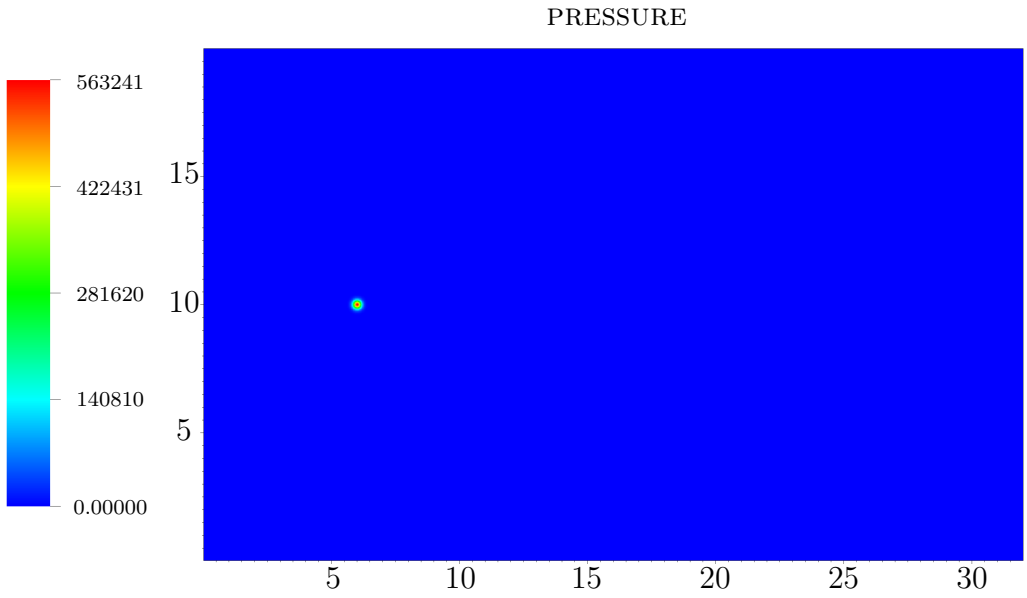

Figure 5.5: The initial condition for the problem inserted at the third time step. A domain of 32 m × 20 m with a point source is inserted at $(6, 10)$ m of 563241.8 Pa with a Gaussian half-width of 0.15 m. A global time step of 8×10^{-6} s is used and absorbing boundary conditions are used on each edge.

resolution of 959 × 599 cells. An initial point source pulse is inserted at $(6, 10)$ m with a Gaussian half-width of 0.15 m wide and a resulting initial pressure of 563241.8 Pa. Absorbing boundary conditions are used at each edge with a Gaussian half-width of 0.25 m. A time step of 8×10^{-6} s is used, and the simulation run for 8000 time steps. This results in a final time of $t = 6.4 \times 10^{-2}$ s. At this time a line plot over the shock wave from point $(25, 10)$ m to point $(30, 10)$ m is sampled for analysis. For the baseline RK3-WENO5 scheme, the final time step of the example problem can be seen in Figure 5.6.

The RK3-HYBRID scheme was then run on the same problem using an r_c value of 1×10^{-4}. The line plots over the shock waves for both schemes can be seen in Figure 5.7. The difference between the solutions for each scheme is shown in Figure 5.8. Two crude percentages of accuracy lost are calculated by dividing the average and maximum difference between both schemes by the distance between the maximum and minimum values of the RK3-WENO5 scheme. In the case of the RK3-HYBRID scheme, there is an average difference of 0.04% with a maximum difference of 0.36% after 8000 time steps. Therefore, it is apparent that the hybrid scheme with a low r_c value displays adequate accuracy when compared to the much

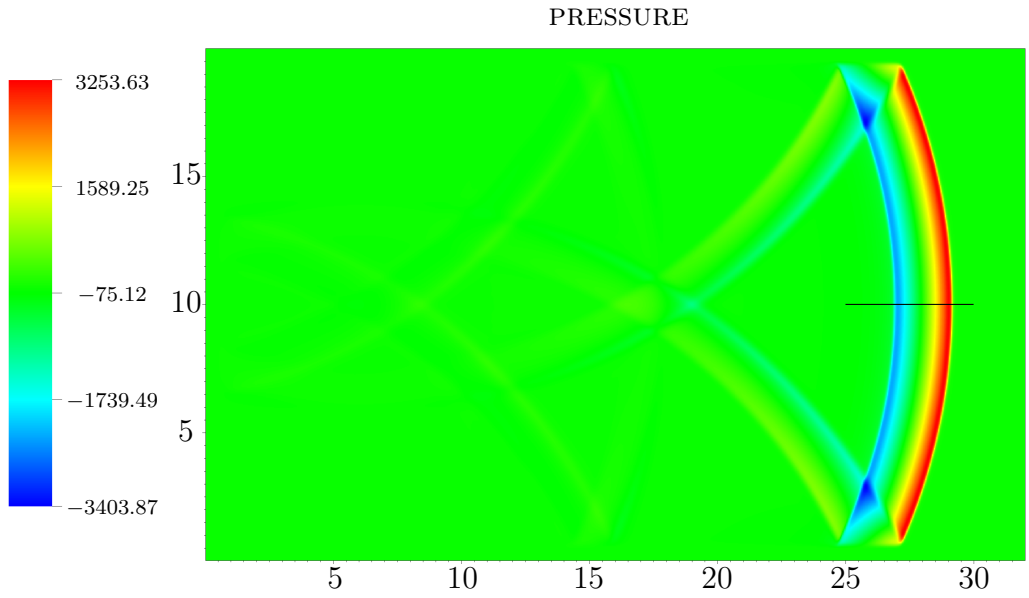

Figure 5.6: The solution at 8000 time steps when $t = 6.4 \times 10^{-2}$ s using the RK3-WENO5 scheme. The black line shows the line in which subsequent plots referring to this example problem are plotted over.

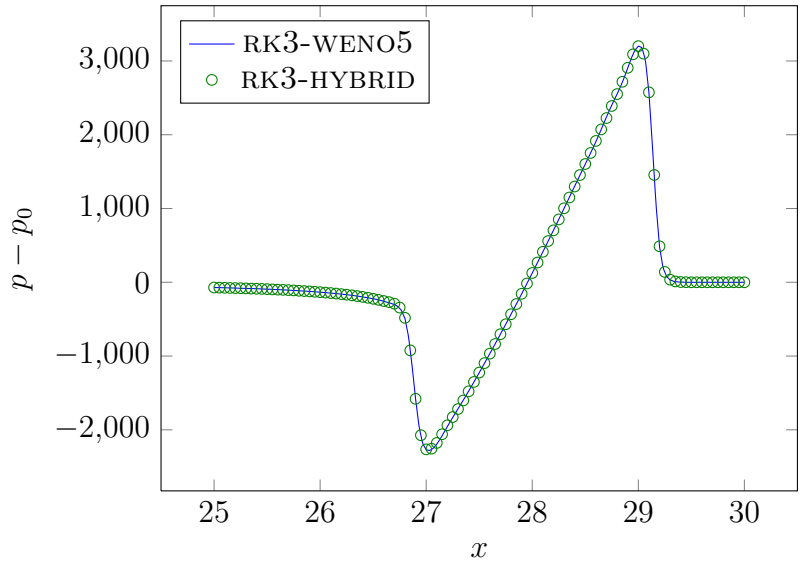

Figure 5.7: Comparison between the RK3-HYBRID and RK3-WENO5 schemes.

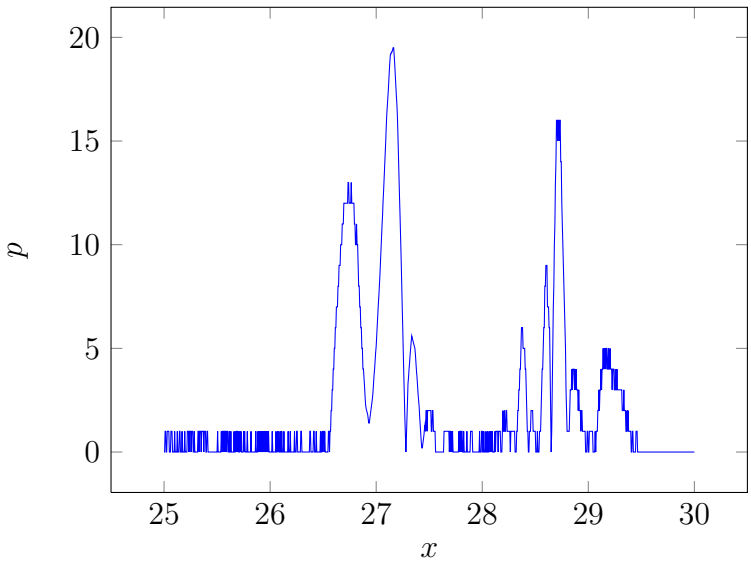

Figure 5.8: The absolute value of the difference between the RK3-HYBRID and RK3-WENO5 schemes.

more computationally intensive WENO scheme. The RK3-WENO5 scheme took 772 minutes to complete, while the RK3-HYBRID scheme took 275 minutes, resulting in only 35.6% of the calculations necessary for the RK3-WENO5 scheme. For reference, a similar example problem with a low amplitude source was run using the RK3-DRP scheme that completed in 231 minutes.

5.5 Verification of AMR Model

In this section the AMR scheme is verified for use in the model. AMR brings with it many more options and increased complexity over control of the model's execution. A simplified approach is taken so as not to focus on the details of the options of AMR. The AMR schemes using one level and two levels of refinement are analyzed. A refinement factor of 2 is used at each level. Therefore, the base grid for the RK3-AMR1 scheme is 479 $\times$ 299 cells and the base grid for the RK3-AMR2 scheme is another factor of 2 smaller.

The solution at 8000 time steps for the RK3-AMR1 scheme is shown in Figure 5.9 along with the patches on each level at that time. Figure 5.10 shows the comparison of shock waves resulting in the RK3-AMR1 and RK3-WENO5 schemes, and Figure 5.11 shows the absolute value of the difference between the schemes. An r_c value used for both the hybrid scheme at the base level and for tagging cells for refinement was 1×10^{-4}. The average difference between the schemes is 0.36% with a maximum difference of 1.39%. The RK3-AMR1 scheme took 262 minutes to complete, resulting in 33.9%

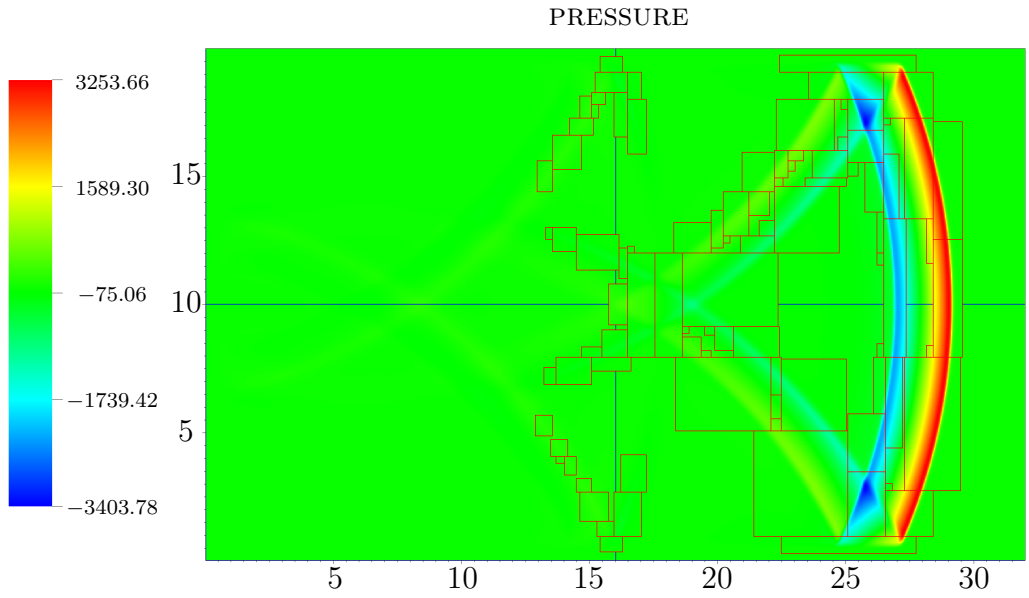

Figure 5.9: The solution at 8000 time steps when $t = 6.4 \times 10^{-2}$ s using the RK3-AMR1 scheme. The blue rectangles are patches on level 0 while the red rectangles are the patches on level 1.

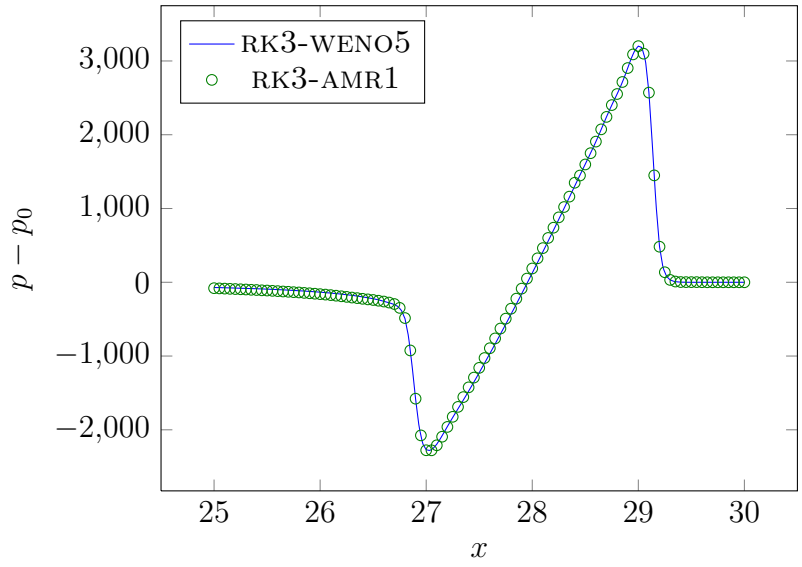

Figure 5.10: Comparison between the RK3-AMR1 and RK3-WENO5 schemes.

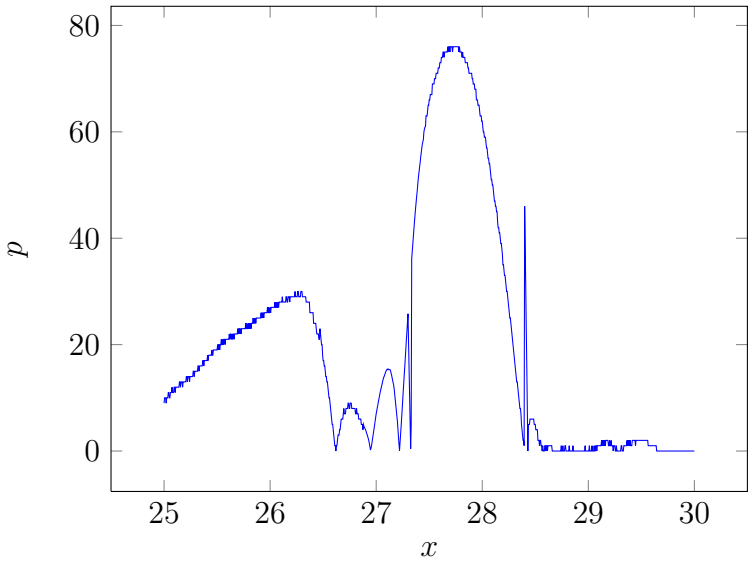

Figure 5.11: Absolute value of the difference between the RK3-AMR1 and RK3-WENO5 schemes.

of the calculations of the RK3-WENO5 scheme. This illustrates a disadvantage to the AMR scheme if the parameters used are too strict. The AMR scheme must do the calculations on overlapping patches, so if a large amount of refinement occurs over much of the domain during the simulation, the performance of AMR can suffer, and with cases where refinement occurs over the entire domain throughout the entire simulation time, performance could theoretically be worse than a static grid approach. In this case it did not perform much faster than the hybrid scheme on a static grid.

The same problem was run using the RK3-AMR2 scheme. A smaller r_c value for the tagging of cells for refinement used was 1×10^{-3} while the r_c value for the hybrid scheme on the base level used was 1×10^{-4}. The solution along with the patches on each level is shown in Figure 5.12. The comparison between the RK3-AMR2 and RK3-WENO5 schemes is shown in Figure 5.13, with the absolute values of the difference between both schemes given in Figure 5.14. The average difference between the two schemes is 0.98% with a maximum of 3.8%. The RK3-AMR2 scheme took 145 minutes to complete, resulting in 18.7% of the calculations of the RK3-WENO5 scheme. In this case, the advantage of the AMR scheme is evident as a small amount of accuracy is sacrificed for a large gain in efficiency. It is also apparent that the AMR scheme is giving adequate results. The AMR scheme's biggest weakness in accuracy is apparent at locations that are not refined, as can be seen between the area of 27.5 m and 28.5 m in Figure 5.14 for example. AMR can also introduce outlying anomalies in the solution as apparent in Figure 5.14 at approximately 28.4 m.

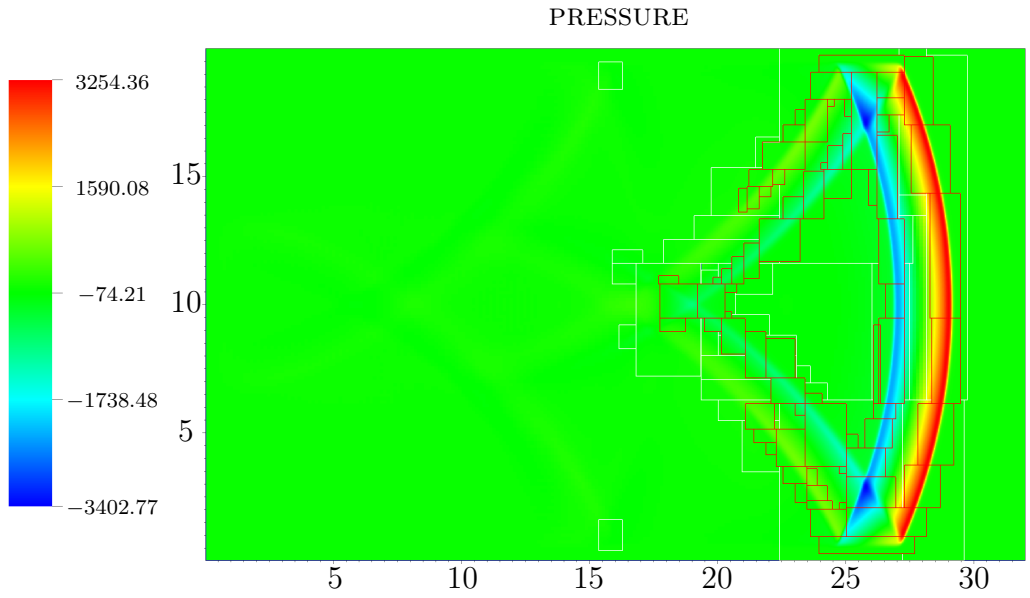

Figure 5.12: The solution at 8000 time steps when $t = 6.4 \times 10^{-2}$ s using the RK3-AMR2 scheme. The blue rectangles are patches on level 0 while the white rectangles are the patches on level 1 and red rectangles are patches on level 2.

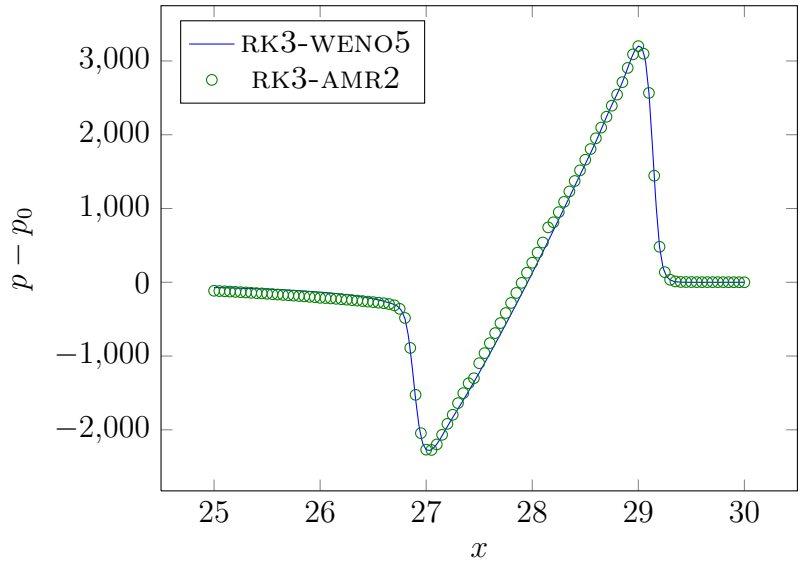

Figure 5.13: Comparison between the RK3-AMR2 and RK3-WENO5 schemes.

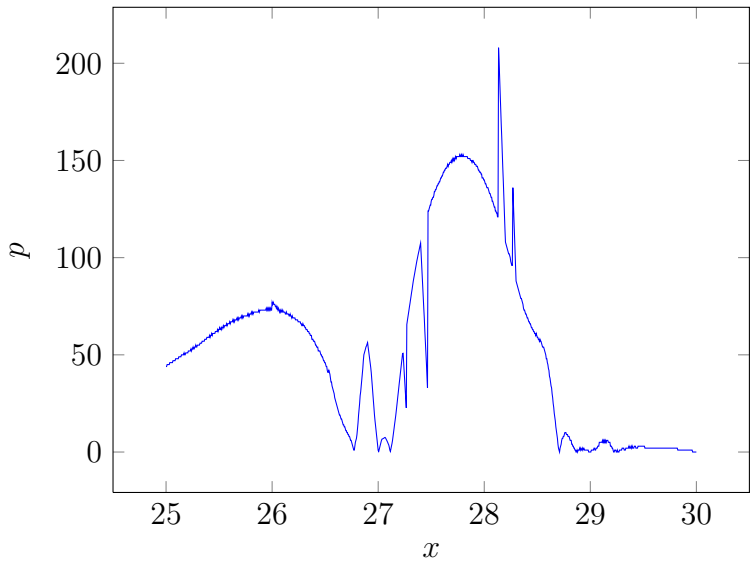

Figure 5.14: Absolute value of the difference between the RK3-AMR2 and RK3-WENO5 schemes.

5.6 Local Lax-Friedrichs Flux Splitting Verification

In this section, a simple verification of the local Lax-Friedrichs flux splitting (LLF) method used in the WENO scheme is done. Wochner suggests a global constant splitting with $300 < \alpha < 400$. However, the standard method is to calculate a global α at each Runge-Kutta stage for use. The AMR implementation benefits from a LLF flux splitting in that a global α value does not need to be synchronized across all patches at every step. The LLF flux splitting is known to violate conservation, but a constant global flux splitting is known to be too diffusive, with the high pressure initial conditions in this work. A large global α would be necessary and result in a lot of unnecessary diffusion at earlier times in the simulation. Therefore, a simple verification that LLF is giving adequate results is done. A constant global flux splitting using an α value of 360—which is not greater than the maximum eigenvalue (about 545 at time 4×10^{-4} s) of the example problem initially—is used and another case with $\alpha = 500$ is also run for comparison.

The results of the RK3-WENO5-G360 scheme can be seen in Figure 5.15 and Figure 5.16. The average difference between the schemes is 0.07% with a maximum difference of 1.22%. The results of the RK3-WENO5-G500 scheme can be seen in Figure 5.17 and Figure 5.18. The average difference between the schemes is 0.13% with a maximum difference of 1.95%.

Although the results of this verification may be significant to other areas of study,

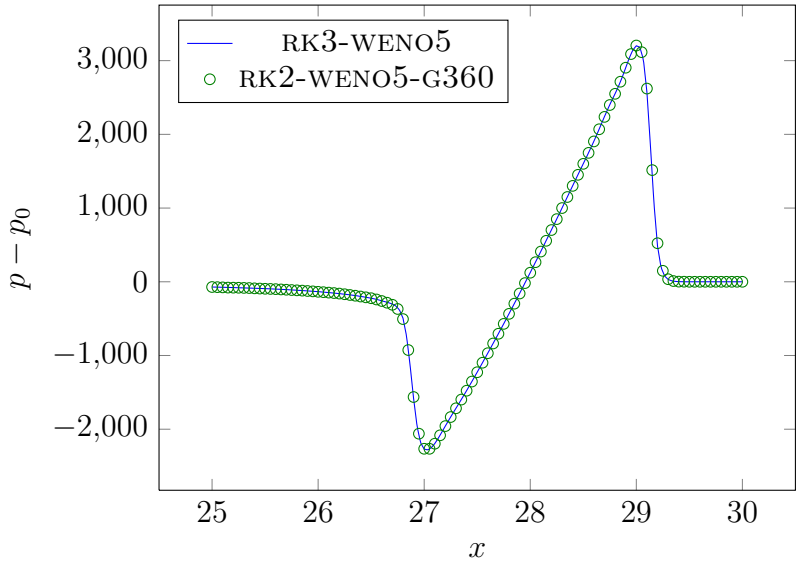

Figure 5.15: Comparison between the RK3-WENO5 and RK3-WENO5-G360 schemes.

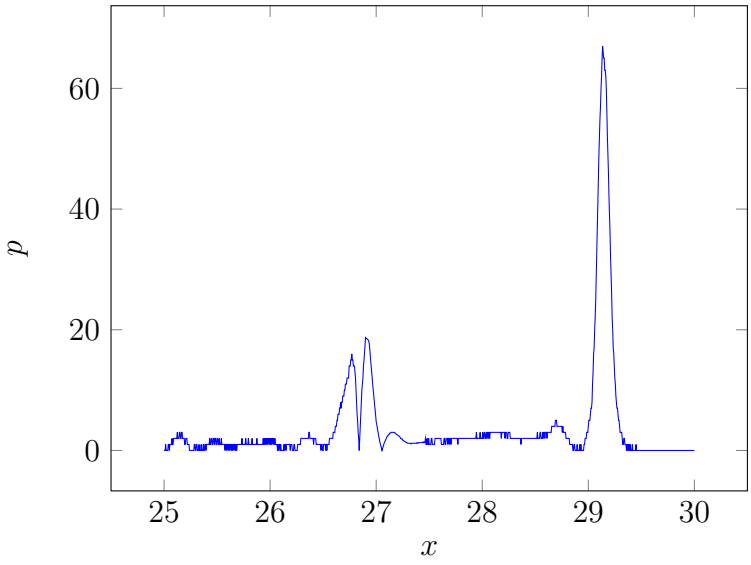

Figure 5.16: Absolute value of the difference between the RK3-WENO5 and RK3-WENO5-G360 schemes.

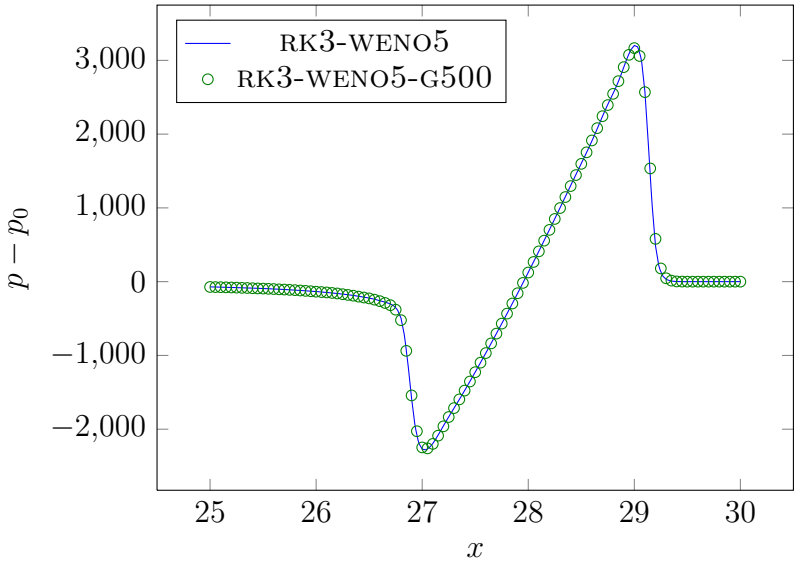

Figure 5.17: Comparison between the RK3-WENO5 and RK3-WENO5-G500 schemes.

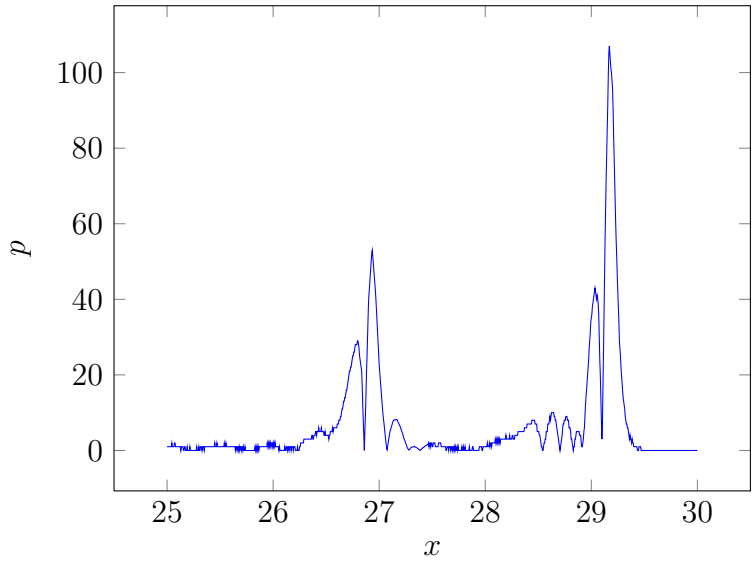

Figure 5.18: Absolute value of the difference between the RK3-WENO5 and RK3-WENO5-G500 schemes.

LLF flux splitting does not appear to deviate enough to warrant avoiding its use in this work. Analytic solutions do not exist for this problem which makes it difficult to know which type of flux splitting is giving more accurate solutions in this case, but what has been inspected here is that by using LLF, the results are acceptable for this work. It is apparent that flux splitting is not given much focus when using high-order schemes in other works that have been investigated during this study and further investigation into flux splitting is not conducted.

5.7 Method Comparison

When using the model on a large scale, accuracy and efficiency of solving the model is a crucial aspect to achieving the desired results. Typically a tradeoff between accuracy and efficiency is necessary. This section specifically analyzes this tradeoff, as the tradeoff between accuracy and efficiency does not necessarily occur in a linear fashion. The main decisions to be made are whether or not to use a WENO3 or WENO5 subscheme in the hybrid and AMR schemes and whether to use a third-order or second-order Runge-Kutta scheme.

5.7.1 Fifth-Order vs. Third-Order WENO

Running the example problem using the RK3-WENO3 scheme results in the comparison against the RK3-WENO5 scheme as shown in Figure 5.19 and Figure 5.20.

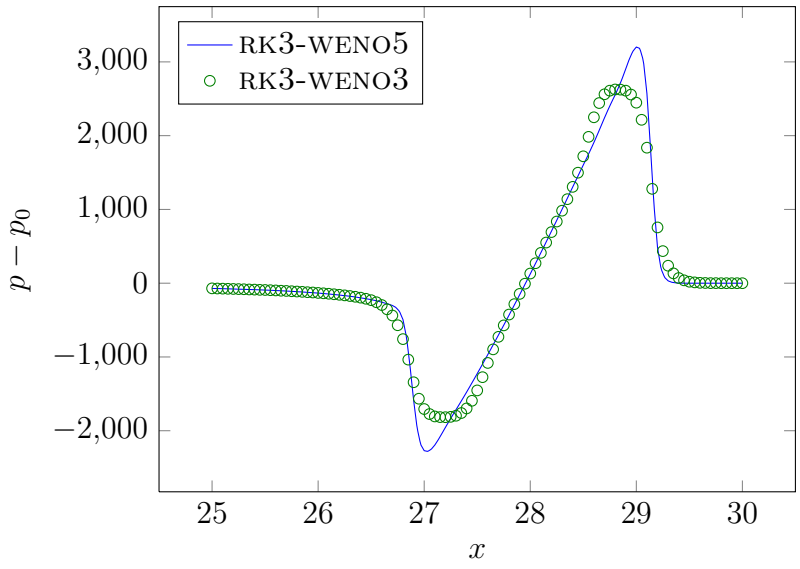

Figure 5.19: Comparison between the RK3-WENO3 and RK3-WENO5 schemes.

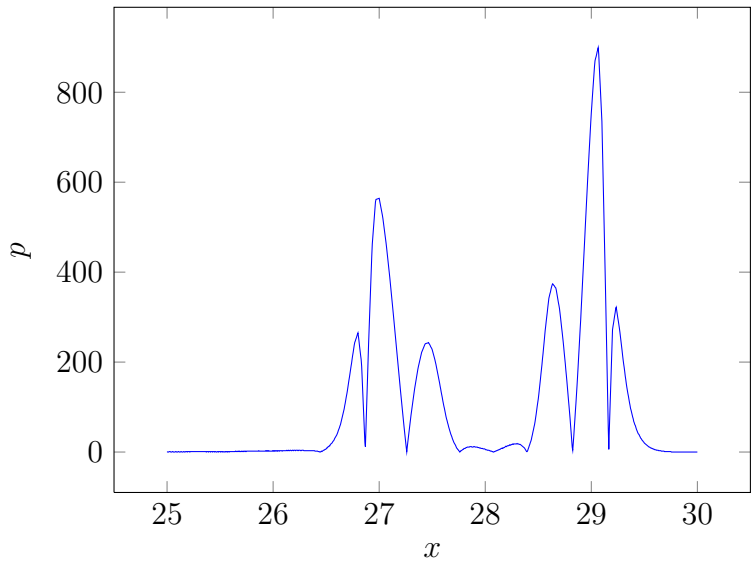

Figure 5.20: Absolute value of the difference between the RK3-WENO3 and RK3-WENO5 schemes.

The average difference between both schemes is 2.0%, but the maximum difference is 16.4% and it is apparent from Figure 5.19 that the RK3-WENO3 scheme does a poor job of resolving the sharp peaks of the shock wave. The RK3-WENO3 scheme took 612 minutes to complete resulting in 79.2% of the calculations of the RK3-WENO5 scheme. In this case, the higher efficiency of the WENO3 scheme does not counteract the loss in accuracy enough to justify its use. Therefore, the WENO3 scheme is avoided for use as a subscheme in any of the other schemes.

5.7.2 Second-Order vs. Third-Order Runge-Kutta

In this section the RK2 and RK3 schemes are analyzed for both performance and accuracy. The example problem was run using both the RK2-WENO5 and RK3-WENO5 schemes. The comparison between the resulting shock waves of both is shown in Figure 5.21 with the absolute value of the differences shown in Figure 5.22.

The average difference is 0.02% with a maximum difference of 0.24%. The RK2-WENO5 scheme took 492 minutes to complete, resulting in 63.7% of the calculations of the RK3-WENO5 scheme. Therefore, in this case the accuracy to efficiency tradeoff is desirable for large simulations and the RK2 scheme is chosen for use in the large simulations in later sections.

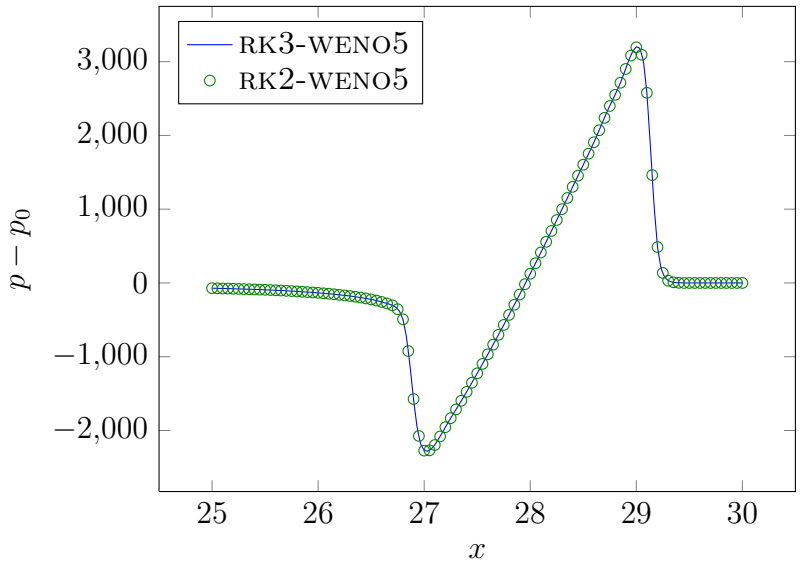

Figure 5.21: Comparison between the RK2 and RK3 schemes using the WENO5 scheme.

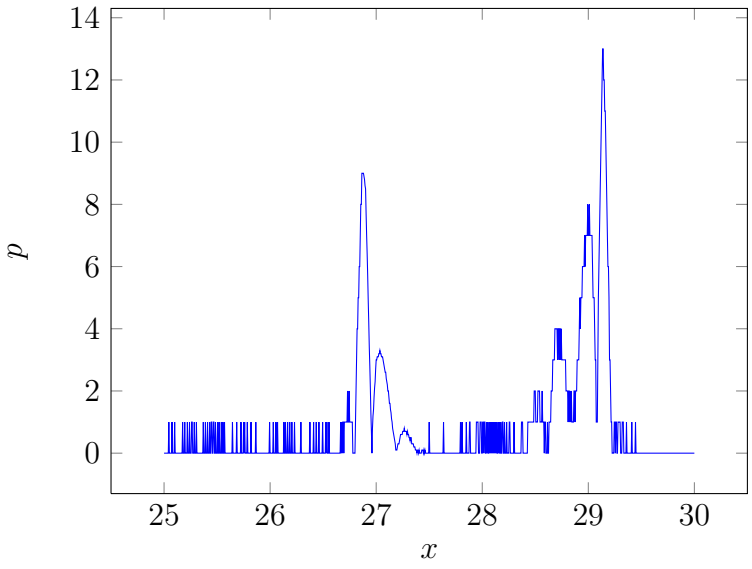

Figure 5.22: Absolute value of the difference between the RK2 and RK3 schemes using the WENO5 scheme.

5.8 Accuracy vs. Efficiency Summary

A summary of the explored method combinations used on the example problem is given in the plots in this section. The data was recorded after 8000 time steps with a time step size of 8×10^{-6} s, corresponding to a CFL of 8.232×10^{-2}. A plot of the accuracy vs. efficiency for the schemes using the mean difference for each scheme is given in Figure 5.23. A plot of the accuracy vs. efficiency for the schemes using the maximum difference for each scheme is given in Figure 5.23. As is evident in these

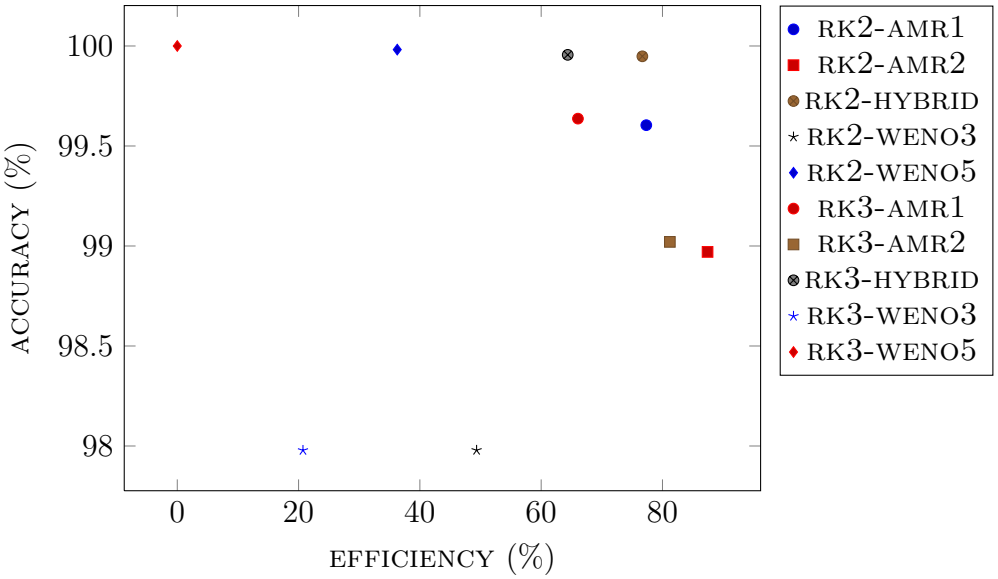

Figure 5.23: Plot of the general accuracy vs. efficiency of each scheme using the mean difference between the RK3-WENO5 scheme. The data from the example problem in this section was used for this plot which was recorded after 8000 time steps.

plots, the RK2-AMR2 scheme appears to be the most efficient while maintaining a moderate accuracy.

Naturally, accuracy of the listed methods decreases in some fashion at each time step, and therefore care must be taken to choose the correct method according to the requirements of the problem. As the methods become more complex in design, the amount of options when using the methods increases, as well as the amount of parameters that can be tuned for the best results. The amount of tunable parameters is enough that a full treatment of them cannot faithfully be conducted in this work. Although the hybrid scheme typically just needs an r_c value low enough to avoid oscillations throughout the simulation, the AMR scheme introduces many more options and combinations thereof, almost all of which are tuned through trial and error as there are no concrete methods for choosing them.

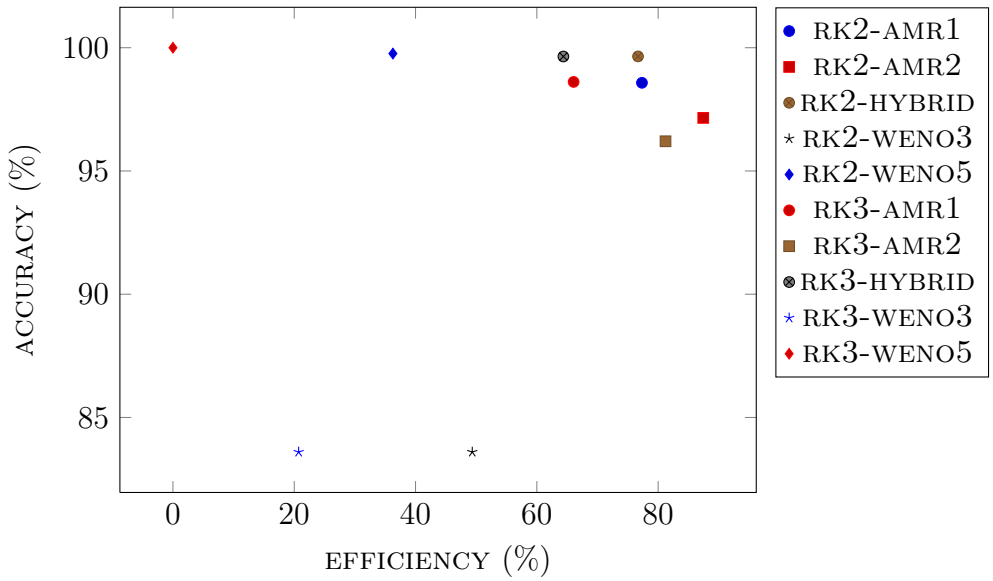

Figure 5.24: Plot of the general accuracy vs. efficiency of each scheme using the maximum difference between the RK3-WENO5 scheme. The data from the example problem in this section was used for this plot which was recorded after 8000 time steps.

5.9 Phenomenological Study

When applying the model as intended, the results obtained do not directly mimic a natural environment in which thunder ordinarily exists. In reality, the signal encounters many physical obstacles and impedances such as trees, hills, and buildings on its path to an observer which result in a wildly variable outcome. Reverberations, echoes, and reflections due to these obstacles all have a major impact on how the signal is ultimately perceived. Signals obtained from the simulations of thunder here are considered *dry* in the sense that they were recorded in an environment that lacks reverberative properties. Although the effects of propagating thunder through obstacles such as buildings and trees, for example, are too complex to be simulated directly and specifically, the recorded signal from the simulation can be post-processed to include the general forms of reverberations and echoes that it would likely experience during its travel due to these obstacles. Several audio editors exist that are capable of adding reverb and echo effects to mimic a signal traveling from a distant source. In this work Adobe® Audition® CS 5.5 was the program used to process the thunder signals simulated in order to establish their validity through a subjective sense. One problem with validating the outcome of the model phenomenologically is that test cases can not be recreated experimentally and compared. Also, thunder can sound quite different according to many factors, such as the length of the channel, width of the channel, power output, geometry of the channel along with branches, amount of restrikes, distance from observer, temperature gradients, wind gradients, obstacles,

etc. Therefore, validating the model audibly is a subjective process and it is also not possible to do so in document form. Nevertheless, there are a few ways in which signals can be compared through the use of spectral and statistical analysis in order to establish their validity. These methods will be employed in this section on three separate thunder events that were generated by the model in the computer and compared to a recording of a natural thunder event. Although the model is capable of handling events with multiple strikes, only single strike events are of focus, due in minor part to the uncertain extreme conditions near the channel that the model may not be able to account for such as extreme heat, but more so due to lack of computational resources. Unfortunately for this work, but fortunately for all research, access to high performance computer resources is limited for reasons of fairness to all research, and therefore the results for this phenomenological study are barred by this limitation in some regard. The test cases here have been designed to fit within the certain amount of CPU hours allocated for the research and the results extrapolated to judge the amount of resources needed for adequate simulation of this phenomena using the model.

To play back the signal that would be perceived at any particular location, it must be recorded. To capture the pressure waves at particular locations, virtual microphones can be placed in cells that contain that location of the domain. A complexity arises when higher level grids propagate to the domain locations of the microphones and there are multiple cells in the hierarchy to record pressure values from. To handle this, the values of pressure at each location and at each level in the hierarchy were recorded at each time step along with which level they were recorded from. The waveform over time could then be post-processed to either use the pressure values at the highest level of the hierarchy, or the lowest. When comparing the results, it was found that using the values at the lowest level of the hierarchy gave similar results when compared to the values recorded at the highest level. The results at the lowest level still captured the higher frequencies that the higher levels carried. This is due to the projection operation used in the AMR scheme where the solutions at the higher levels are copied downward. The pressure values in the cells are allowed to change at a frequency dictated by the time step size. As long as the time step size allows for capturing frequencies higher in time than the highest level can capture in space, the base grid is then capable of the capturing frequencies according to the highest level of the hierarchy. Therefore, values recorded at the base grid are used in the results. The time step sizes used are of the same sample rate as standard wave files used by the computer, e.g. 1/22050 s, 1/44100 s, etc.

An actual recording of thunder is also used in this section which was recorded at an unknown length from the strike. For reference, Figure 5.25 shows the waveform of the first two seconds of this recording.

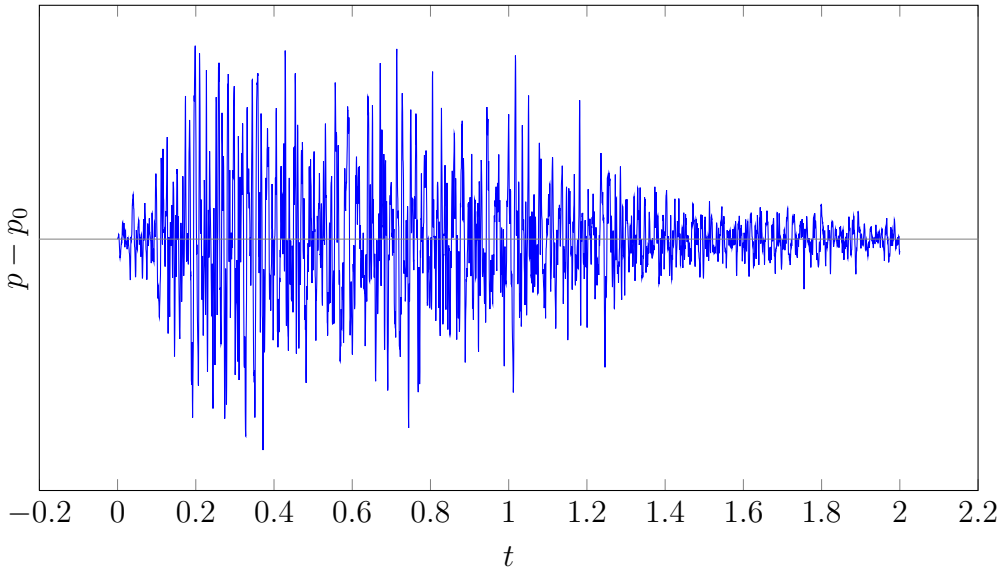

Figure 5.25: The first two seconds of an actual recording of thunder.

5.9.1 Event Example 1

The first thunder event was run using 360 processes which completed in 535 minutes. The domain size was 2 km × 2 km. A resemblance of the geometry of the lightning channel can be seen in the first frame of the pressure profiles in Figure 5.26 in which the waves have only propagated a short distance. The pressure profile of the entire domain at other times can also be seen in Figure 5.26. A base level grid of 4999 × 4999 cells was used with two levels of refinement, each using a refinement factor of two. This gives cell sizes of 0.4 m on the base grid and 0.1 m on the highest level grid. The corresponding wavelength capturing ability according to 10 PPW rolloff is then 85.75 Hz on the base grid and 343 Hz on the highest level grid. Lightning channel segment lengths of 6 m, along with a channel width of ∼12.5 cm were used. An initial instantaneous density source was inserted so as to maximize the frequency of the initial waves since the energy input into the channel occurs within microseconds, resulting in an initial pressure pulse of ∼10 atm. Virtual microphones were placed at locations $(500, 10)$ m and $(1500, 10)$ m. The time step size used was 1/22050 s which allows for frequencies of up to 11025 Hz to be recorded in time. The waveforms recorded over the 8.5 s of simulation time can be seen in Figure 5.27. These waveforms were then processed to include reverb and echo. A post-processed waveform for Mic 2 can be seen in Figure 5.28. Audibly, the post-processed waveforms resemble the sound of thunder (excluding the "peel" that the stepped leader plays a large role in), although they do lack some higher frequency features apparent in the recording of actual thunder. Even by visibly comparing the waveform of Figure 5.28 to Figure 5.25, it is apparent that Figure 5.28 would appear more similar if it was squeezed, i.e. pitch

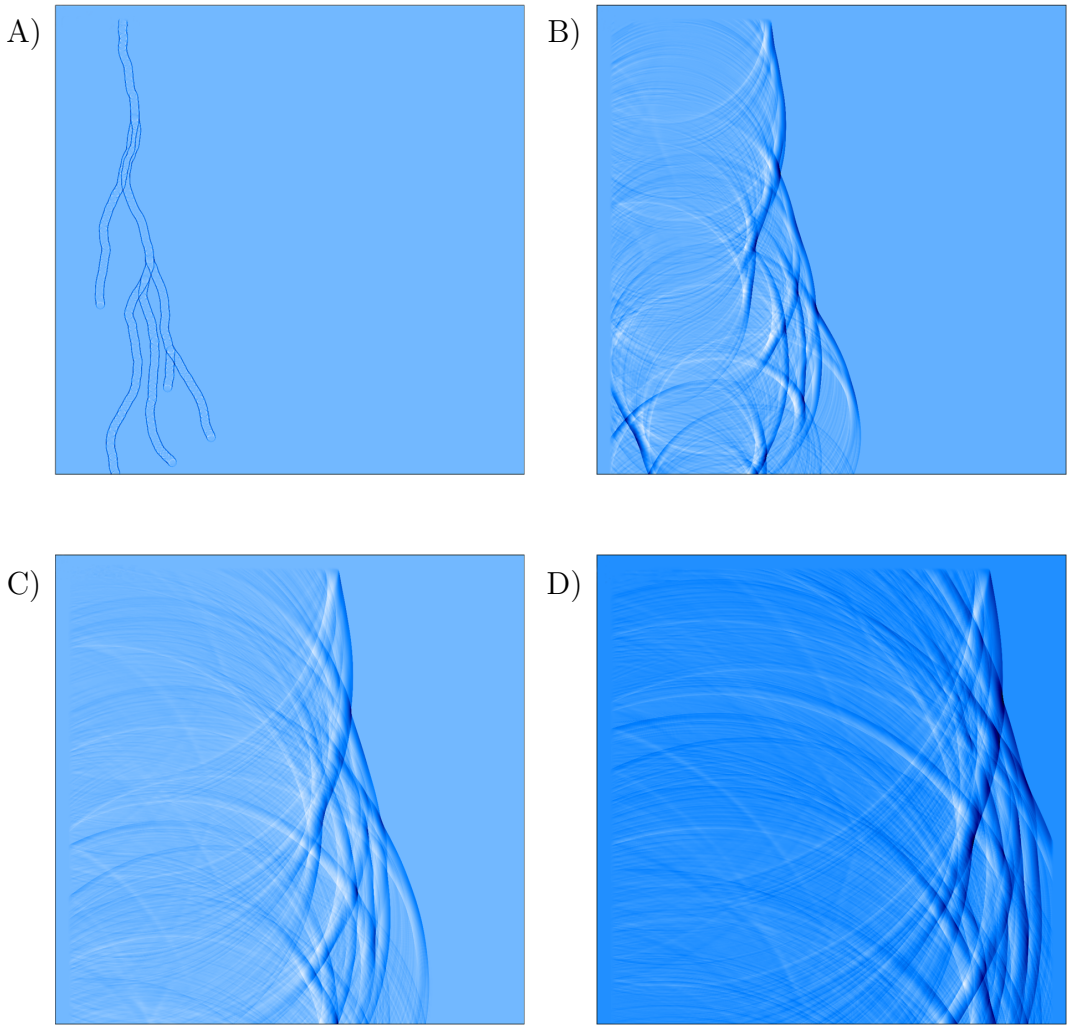

Figure 5.26: Pressure profiles of Event 1. A) $t = 0.0453515$ s. B) $t = 1.31519$ s. C) $t = 2.67574$ s. D) $t = 4.03628$ s. Darker colors are high pressure, with lighter colors being low pressure.

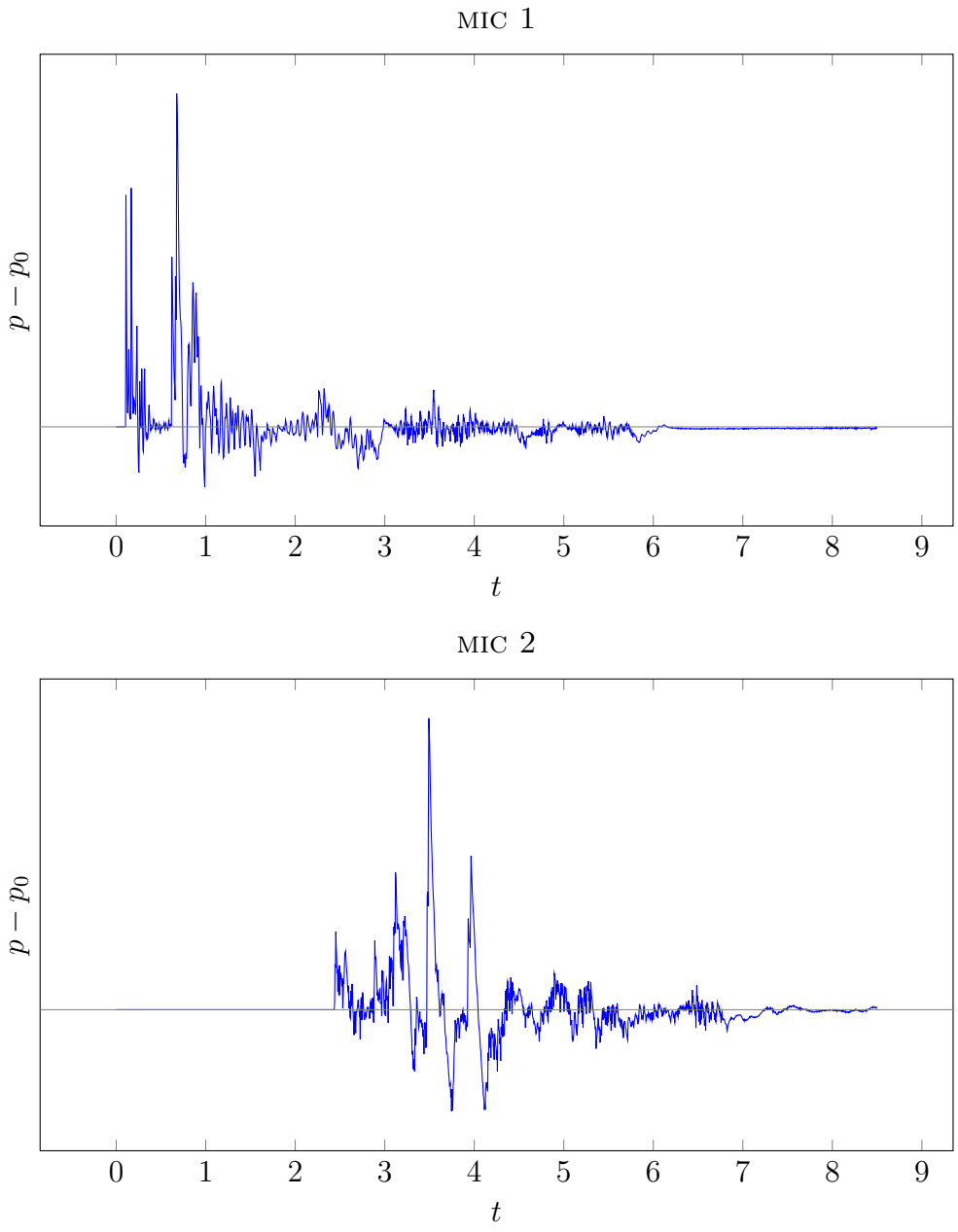

Figure 5.27: Two virtual microphone recordings from Event 1. Mic 1 is located at $(500, 10)$ m and Mic 2 at $(1000, 10)$ m. Note that Mic 1 is located to the left of a lightning branch and experiences left traveling waves early on.

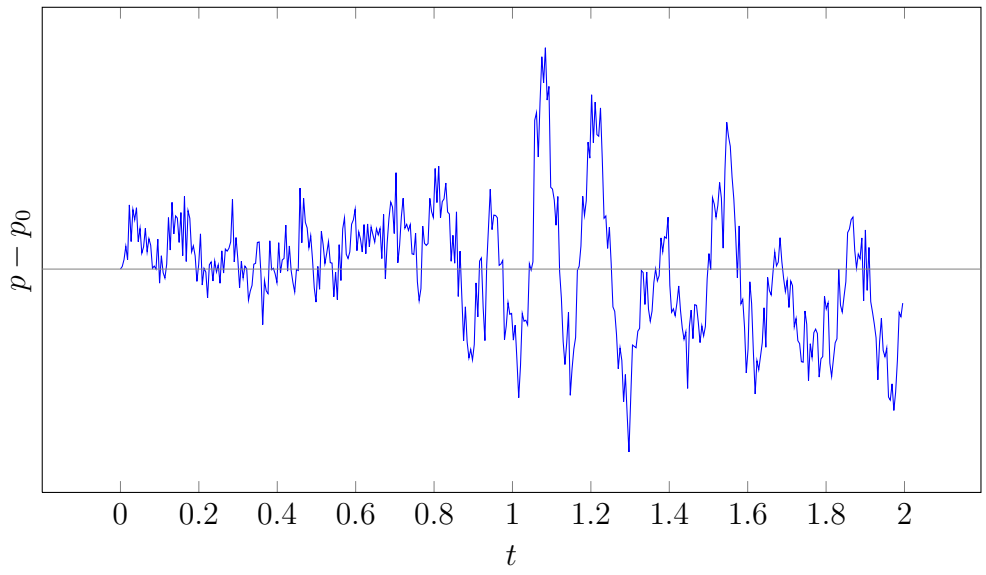

Figure 5.28: The first two seconds of the post-processed recording of thunder from Mic 2 in Event 1.

shifted upward. However, visual similarity between waveforms is subjective and does not necessarily prove a genuine similarity.

Spectral analysis was performed on the first two seconds of samples in which pressure deviates from ambient pressure in both the simulated thunder and the recording of actual thunder. This spectral analysis can be seen in Figure 5.29. A fast fourier transform (FFT) of size 16384, using a Blackmann-Harris window was used for the spectral analysis. The peak power for the actual recording of thunder exists around 80 Hz as is typically the case in actual thunder. The simulated thunder, however, peaks at very low frequencies which is likely unnatural. Two factors that go hand-in-hand and are likely affecting this result are the resolutions of the grid hierarchy and the width of the source channel. It is likely that the channel width plays a key role in the resulting spectral density of the synthetic thunder.

Another test that would have increasing meaning as more recordings of results for both synthetic and actual thunder are available to add to the data set is the comparison of the distribution of the amplitudes from each signal. This comparison is included here with the knowledge that the amount of data obtained for both types of signals (real and synthetic) is small enough that more data could alter the conclusions. The first two seconds of each signal (the synthetic signal used was the post-processed version) were sampled and the distribution of the amplitudes plotted as a histogram. The histograms of each signal can be seen in Figure 5.30. The waveforms were normalized to scale each of the distributions for comparison. The real thunder resembles a Cauchy distribution which favors small amplitudes relative to large amplitudes. The synthetic thunder is shown to favor larger amplitudes relative

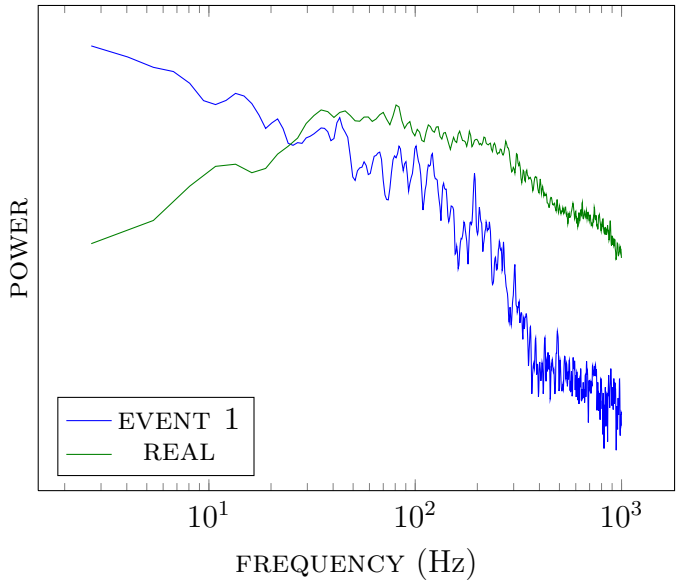

Figure 5.29: A plot of the spectral density for Event 1 using Mic 2. The blue graph is taken from a sampling of the first two seconds of the computer generated thunder while the green graph is a sampling of the first two seconds of an actual thunder recording.

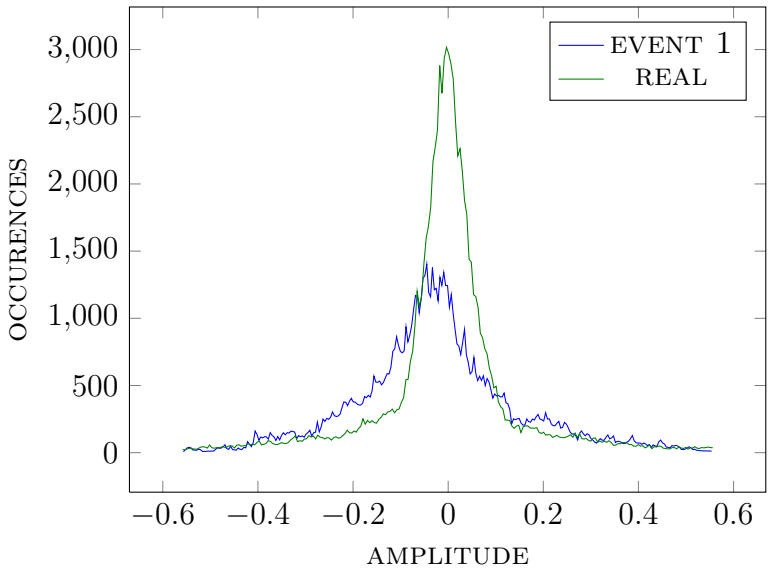

Figure 5.30: A histogram of the amplitudes of the first five seconds of thunder. The blue graph is the result for Event 1 using Mic 2 and the green graph is the result for real thunder.

to the real thunder signal. The likely explanation for this is that the synthetic thunder has been simulated using parameters that are shifting the characteristic frequency to lower than natural values which creates longer lasting high amplitude fluctuations, as well as low frequency waves having lower attenuation over distances than higher frequency waves.

5.9.2 Event Example 2

The second thunder event was run using all the same parameters as Event 1, but with a different lightning channel geometry in order to verify that it incurs similar results as Event 1. The simulation was run using 300 processes and completed in 634 minutes. The geometry and pressure profiles at certain times can be seen in Figure 5.31. The waveforms recorded at locations $(500, 10)$ m and $(1500, 10)$ m can be seen in Figure 5.32.

The spectral analysis on the second thunder event compared to the same actual thunder recording as given in the first thunder event is shown in Figure 5.33. Again, a similar conclusion is reached as with Event 1 and although the post-processed waveform audibly resembles real thunder, the frequencies are shifted too far to the left to appear natural. Similar results obtained for Event 1 according to the distribution of amplitudes are also obtained for this second event as shown in Figure 5.34 that corroborate unnaturally low frequencies.

5.9.3 Event Example 3

As is apparent in the first two simulated thunder events, the width of the source channel appears to play a crucial role in the spectral density of the resulting signal and the resolution of the grid hierarchy plays the role in allowing channels of small widths. Therefore, a third event was run on a smaller domain of 1 km $\times$ 1 km with a higher resolution, using a base grid of 4999×4999 cells, a level 1 refinement factor of 3 and a level 2 refinement factor of 2, which results in an effective grid size of $\sim 30000^2$ cells. The base grid is able to resolve frequencies in space of 171.5 Hz and the highest level is able to resolve frequencies of 1029 Hz. A time step size of 1/44100 s is used in this case as a time step of 1/22050 s proved unstable. Lightning channel segment lengths of 3 m and a channel width of $\sim$6 cm are used. A virtual microphone was placed at $(900, 10)$ m and recorded for 5 s of simulation time. The simulation was run using 300 processes and completed in 701 minutes. A spectral analysis of the first two seconds of when the thunder signal first arrives at the microphone is shown in Figure 5.35, along with the actual recorded thunder. In the figure, it can now be seen that the simulated thunder peaks at $\sim$6 Hz and it can be assumed that as the width of the lightning channel moves to smaller values, the closer the characteristic frequency of the signal can get to the natural range of 40–100 Hz.

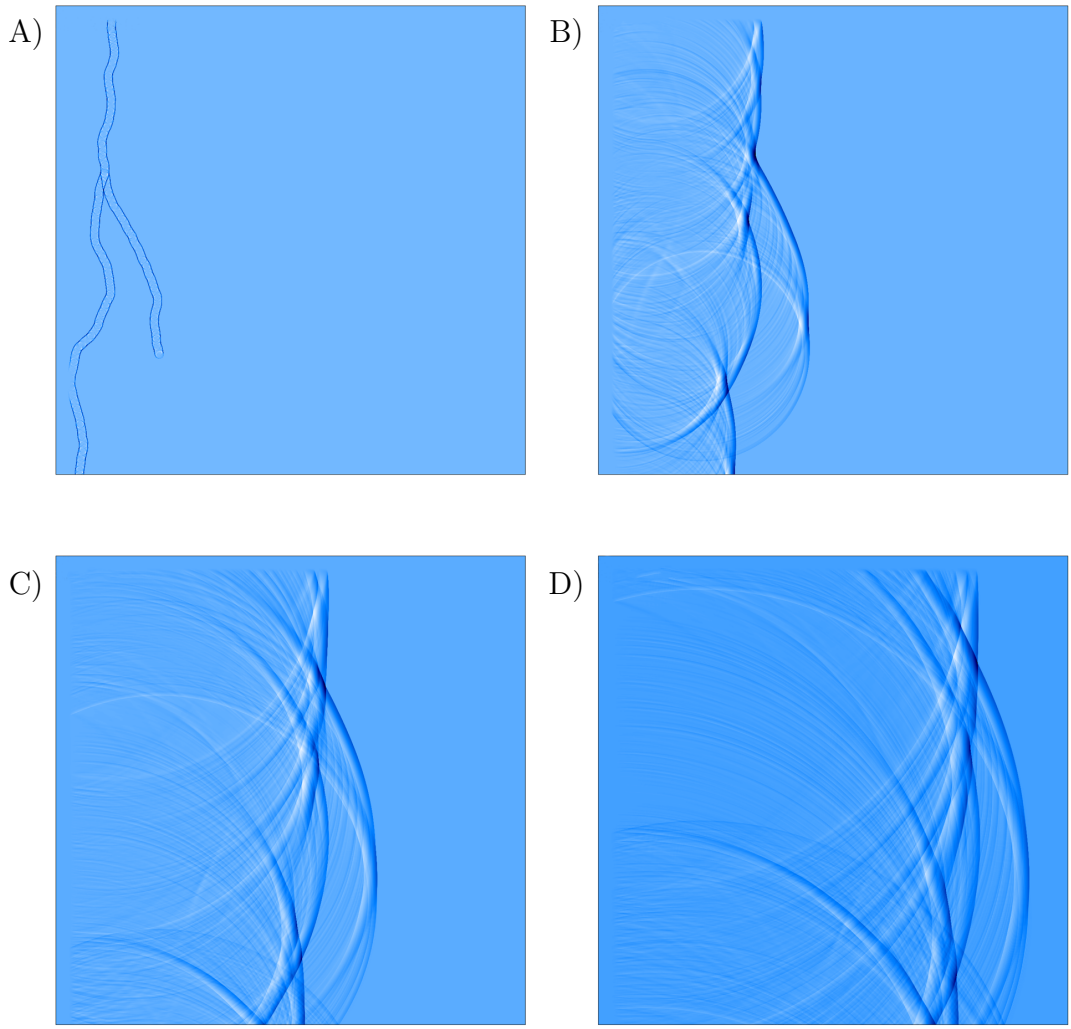

Figure 5.31: Pressure profiles of Event 2. A) $t = 0.0453515$ s. B) $t = 1.31519$ s. C) $t = 2.67574$ s. D) $t = 4.03628$ s. Darker colors are high pressure, with lighter colors being low pressure.

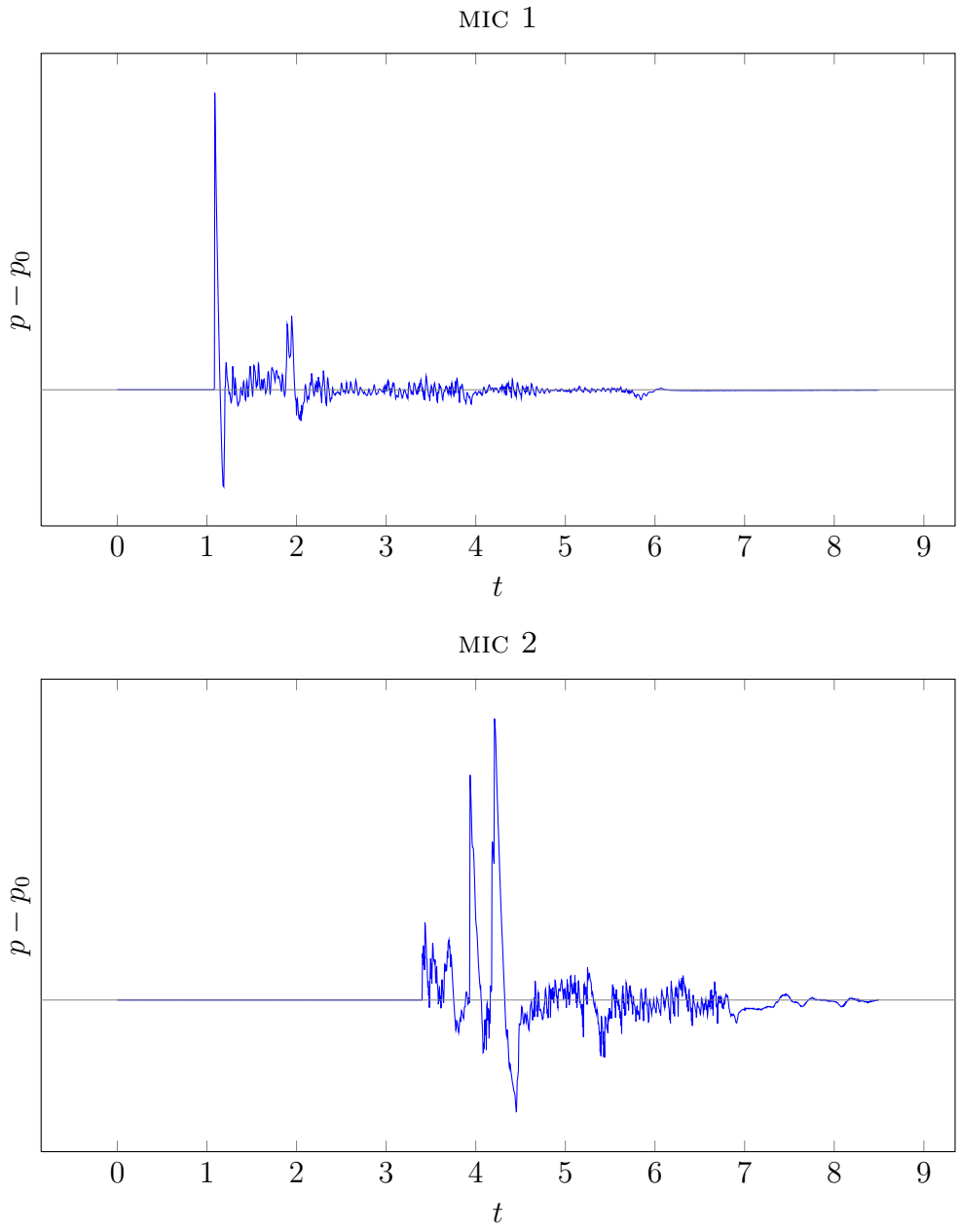

Figure 5.32: Two virtual microphone recordings from Event 2. Mic 1 is located at $(500, 10)$ m and Mic 2 at $(1000, 10)$ m.

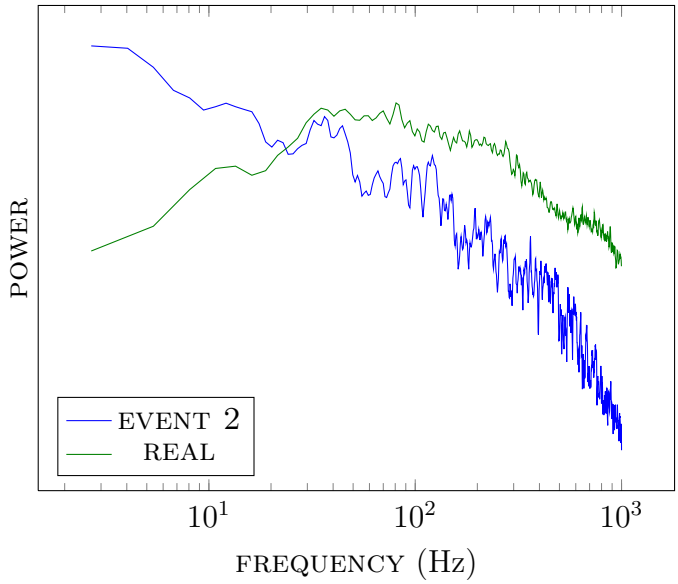

Figure 5.33: A plot of the spectral density for Event 2 using Mic 2. The blue graph is taken from a sampling of the first two seconds of the computer generated thunder while the green graph is a sampling of the first two seconds of an actual thunder recording.

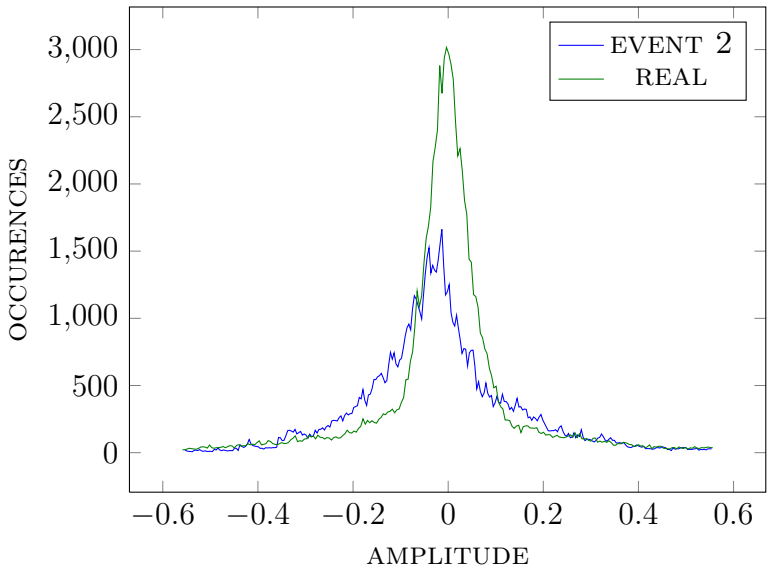

Figure 5.34: A histogram of the amplitudes of the first five seconds of thunder. The blue graph is the result for Event 2 using Mic 2 and the green graph is the result for real thunder.

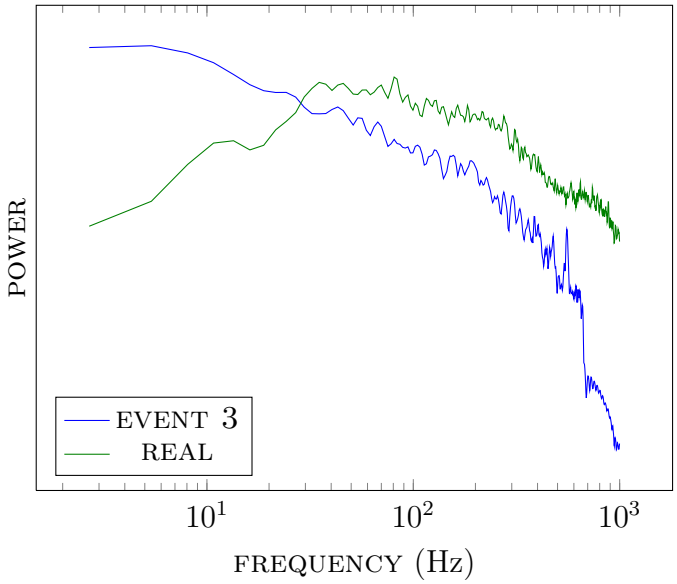

Figure 5.35: A plot of the spectral density for Event 3 using Mic 2. The blue graph is taken from a sampling of the first two seconds of the computer generated thunder while the green graph is a sampling of the first two seconds of an actual thunder recording.

5.9.4 Wave Resolution

As mentioned in Chapter 1, a lightning channel's width is known to be about one centimeter in diameter. This source width ultimately has an effect on the characteristic frequency of the thunder's signal. The source width in the simulation should be a size small enough to bring the characteristic frequency of the signal into the range of 40–100 Hz, where over the entire signal for actual thunder, the peak frequency power typically occurs. This means that the cell sizes for the mesh should ideally be small enough to faithfully describe a source of sufficiently small width. However, having cell sizes smaller than one centimeter requires a very large amount of cells to describe a domain that is an overall size on the order of kilometers. This section serves to analyze what parameters a grid hierarchy should have and what computer system should be available in order to faithfully predict actual thunder with this model.

A problem similar to the example problem for comparing methods was run to study the effect that the grid resolution has on the waves. A virtual microphone was used to record the pressure at the location (10, 10) m at each time step. Each point source pulse located at (6,10) m had an initial amplitude of ∼10 atm. A case with a low resolution grid was run using finest grid cell sizes of 0.1 m with a source width of ∼12.5 cm and time step sizes of 1/22050 s. The same problem was then run with a higher resolution grid using finest grid cell sizes of 0.01 m and time step sizes of 1/96000 s. Finally, the problem was run on the high resolution grid using a source width of ∼1 cm. The results can be seen in Figure 5.36. For the ∼1 cm width, 0.01 m

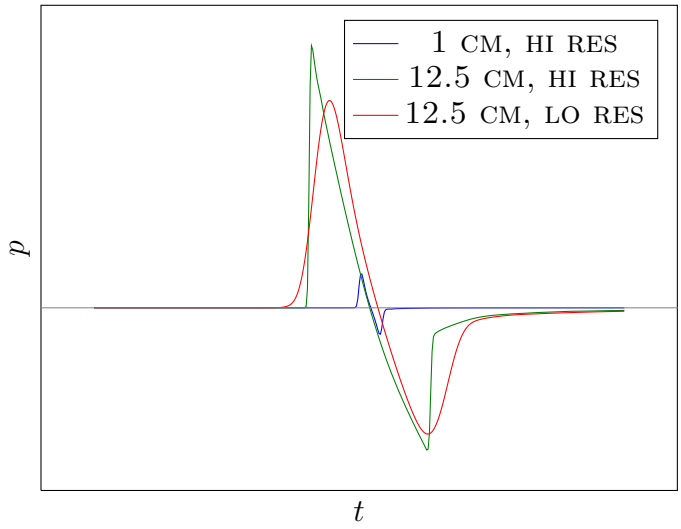

Figure 5.36: A virtual microphone in a domain of 20 m × 20 m was placed at (10, 10) m in the example problem used for comparing methods and run at different grid resolutions and source widths. Each point source pulse located at (6,10) m had an initial amplitude of ∼10 atm. The red graph is the ∼12.5 cm wide source with the width of the finest cells being 0.1 m. The green graph is the same, but with cell sizes of 0.01 m. The blue graph had a source width of ∼1 cm and cell sizes of 0.01 m. The higher rate of absorption on higher frequencies is also apparent in the blue graph. All three waves are given at the same scale.

cell size case, the wavelength is roughly 0.28 m after propagating 4 m and therefore the PPW is 28 in Figure 5.36. For the ∼12.5 cm width, 0.01 m cell sizes case, the PPW is 141 based on a ∼1.41 m wavelength, and shows much more definition in the nonlinearity of the wave. The ∼12.5 cm width, with 0.1 m cell sizes case, the wave has roughly 16 PPW based on a ∼1.67 m wavelength and has a much less sharp head and tail. This shows that for nonlinear waves, the 10 PPW guideline discussed earlier should be higher to capture nonlinear waves. The guideline of 10 PPW still exists, but the reader need only be aware of this property of nonlinear waves to achieve desired results. Figure 5.36 also shows how the tails of the waves predicted by the model have tails that linger at amplitudes below zero, which may explain the bias of the histograms in Figure 5.30 and Figure 5.34 toward negative amplitudes.

It is difficult to decide on the channel width that would bring the characteristic frequency into the range of 40–100 Hz without the computational resources to do so. Therefore this information must be extrapolated from attainable data. To do so, a simple simulation was used involving a very narrow and very long domain using a segment source to propagate waves from a single segment. Four separate cases were run and the wavelengths of the nonlinear waves using sources of both ∼10 atm and ∼100 atm amplitudes with source widths of ∼12.5 cm and ∼1.25 cm were measured at increasing distances (in meters) from the initial pulse. The nonlinear waves of the four

cases stretch in a decaying power law fashion as shown in the graph in Figure 5.37. This sampled data was then fitted to power law type curves as shown in Figure 5.37 as well. These curves are used to extrapolate the frequency of the waves at exceedingly longer distances.[1] In Figure 5.38, it is apparent that the low characteristic frequency

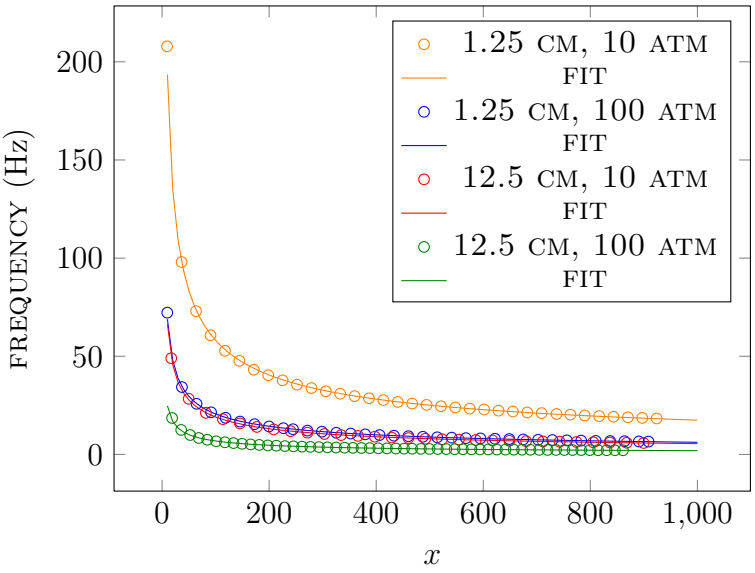

Figure 5.37: A plot of the approximate frequency from nonlinear waves emitted from a segment for four separate cases at increasing distances from the initial pulse. The dots are sampled data. Orange is a ∼1.25 cm width at ∼10 atm. Blue is a ∼1.25 cm width at ∼100 atm. Red is a ∼12.5 cm width at ∼10 atm. Green is a ∼12.5 cm width at ∼100 atm. The sampled sets of data taken from the simulations were then fitted to power law curves which are the corresponding lines.

from Event 1 and Event 2 is corroborated by the ∼12.5 cm, ∼10 atm case, giving single digit frequencies at about the 2 km mark. Interestingly, the ∼1.25 cm width, ∼100 atm case gives roughly the same curve, showing that increasing the pressure by a factor of 10 has about the same effect on wavelength (not frequency, due to faster propagation) as reducing the channel width by a factor of 10. The case of a source width of ∼1.25 cm, with an amplitude of ∼10 atm moves to single digit frequencies at approximately the 8 km mark. For this reason, it appears possible that to achieve a characteristic frequency between 40–100 Hz even around the 5 km mark, the source width may even need to be less than 1 cm, or other factors may be increasing the

[1]The frequency is calculated using the formula of $f = 343/\lambda$. However, the waves would be traveling faster than 343 m/s at initial times, giving an increased frequency, but at later times when they begin traveling close to 343 m/s the graphs are correct. Therefore, the results are showing the wavelength over time and the frequency according to 343 m/s, which is not true at short distances, but it is true at longer distances, which are of concern in this case.

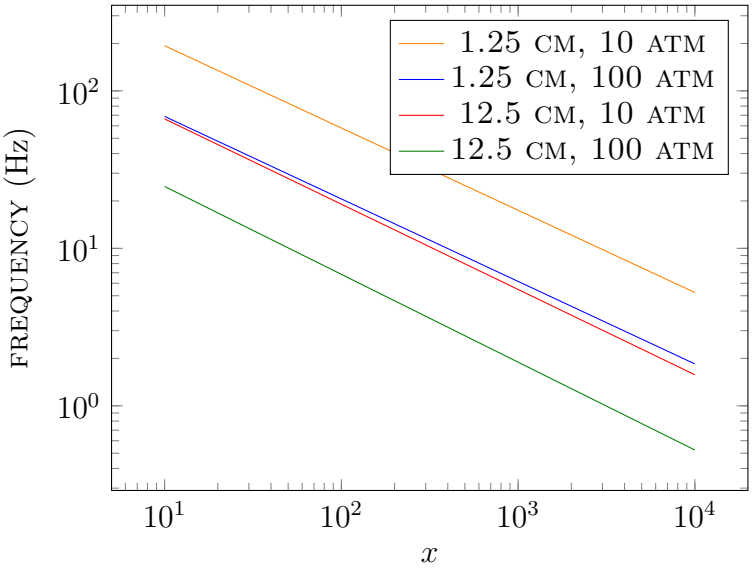

Figure 5.38: A log-log plot of the fitted curves of Figure 5.37 to distances of 10 km.

frequency in actual thunder such as very small scale tortuosity for example, which is difficult to simulate without large computational resources. The highest line in Figure 5.38 would need to be shifted upward by about 30 Hz at the 10 km mark. Assuming any source amplitudes greater than 10 atm may shift the line downward as well. However, the speed of the waves at higher amplitudes may cancel the effect on the stretched wavelengths with respect to the frequency of the waves. Therefore, based on the information in this section, it is apparent that to simulate very near realistic thunder situations with this model, ten to one hundred times the computing power used on the large simulations in this work would be desired to simulate small source widths with several PPW to study the progression to higher characteristic frequencies. With less than 400 processes being used and the simulations completing in around 650 minutes, the necessary computational power is within reach using current supercomputers.

Another consideration that would need to be taken into account using such high resolution grids would be the instability of numerical differentiation used in the model. As cell sizes get smaller, catastrophic cancellation due to finite arithmetic operations in the computer can occur from the differencing of nearly equal quantities in neighboring cells. Possible ways to counteract that situation may be by using Richardson's extrapolation or even foregoing numerical differentiation by switching to a finite volume method for example.

5.10 Summary

In this chapter, the ability to include the effect of refraction into the model through the use of the c term in the pressure equation was described and verified for use. The ability to include the effect of wind through an additional force in the momentum equation was also described and verified for use. The hybrid scheme and AMR schemes were then verified on an example problem and their efficiency and accuracy tradeoffs examined. A simple verification of the local Lax-Friedrichs flux splitting technique was conducted to be sure that results were not too far off from the standard method of global flux splitting since a local flux splitting is preferred for a parallel programming situation. The third-order and fifth-order WENO schemes were compared and the third-order WENO scheme was decidedly expunged for use in any subscheme due to lack of efficiency and lack of accuracy. The third-order Runge-Kutta was compared to the second-order Runge-Kutta and the second-order version was chosen for large simulations as it loses little accuracy while omitting an entire stage of recalculating the right-hand side of the main model equation which results in needing only about 2/3 the calculations of the third-order Runge-Kutta. Summary plots of accuracy vs. efficiency were given and the RK2-AMR2 scheme was chosen as a base method to begin with the large simulations of thunder. A phenomenological study was conducted in which two events were compared against an actual recording of thunder. The main discrepancy between the simulated thunder and actual thunder signals was the spectral peak of the synthetic thunder being in the 1 Hz range which is unnatural as the real thunder generally peaks between 40–100 Hz. A third simulation was run to ensure that decreasing the width of the lightning channel would serve to shift the peak power to the right. Audibly, the post-processed synthetic thunder resembles real thunder, but lacks the initial "peel" typically heard close to a lightning strike. To obtain good physically accurate results, the grid hierarchy resolution and the width of the channel should be small enough to shift the peak frequency to a range of 40–100 Hz. To do so would require between ten to one hundred times the computational power used in the large event simulations in this work. Current supercomputer hardware would hold no limitation in this respect, although a possible loss of significance from the use of numerical differentiation on such high resolution grids may. As seen in Section 5.9.4, the width of the channel also has a significant effect on the attenuation of the signal over distance.

Chapter 6

Conclusion

6.1 Summary

Previous methods for generating synthetic thunder in the computer were overly simplified due in large part to computational constraints. Such computational constraints did not allow for complex models to be solved through finite difference (or other computationally intensive) methods. As computational power has increased, so has the development of more useful and exacting models and methods for simulating physical phenomena. One area in which this has occurred in is nonlinear acoustics and they had not yet been applied to simulating the physical phenomena of thunder.

A general physically-based model was chosen for modification and utilization in simulating the physical phenomena of thunder which occurs as a direct result of lightning. This model includes the effects of geometrical spreading (2D), modified classical absorption, molecular relaxation due to nitrogen and oxygen, finite-amplitude (nonlinear) waves, and wave interaction, as well as the ability to handle multiple dimensions. The model was modified to remove molecular relaxation due to oxygen, as it increases the restriction on the size of the time step necessary and its effects are not very pronounced at the low frequency nature of thunder.

This model is typically chosen to be solved through the use of a weighted essentially non oscillatory (WENO) scheme or a dispersion-relation preserving (DRP) scheme. The WENO scheme is used to the advantage of successfully capturing and propagating nonlinear waves in a stable manner. The disadvantage to the WENO scheme is its computational complexity and intensity. On the other hand, the DRP scheme needs significantly less computations in order to arrive at a solution nearly as accurate as the WENO scheme. The disadvantage to the DRP scheme, however, is that it exhibits unphysical oscillations at discontinuities. For the reason of these advantages and disadvantages of both schemes, a hybrid scheme is introduced to use both schemes in coalition to solve the model and take advantage of the strengths of both schemes. To do so, the hybrid scheme involves the calculation of a smoothness indicator over

the entire domain to decide the areas in which to apply the particular subscheme. Although this extra computation is necessary for the hybrid scheme, it does not counteract the performance gain of using the hybrid scheme.

To solve the model over a large domain and a high range of scales necessary to simulate the process of thunder, another technique known as adaptive mesh refinement (AMR) is also implemented in the solution of the model. This technique exists in a few forms, but in general it uses grids in which coarse cells can be refined into smaller cells in order to capture finer features which would not be possible in a static grid situation without increasing the amount of cells in the entire static grid by shrinking them to the cell size necessary to capture the finer features desired. Using the technique of AMR is very complicated and was therefore implemented through the use of a software package developed for the task known as SAMRAI. When implementing AMR correctly, it can typically achieve efficiency gains in the model execution close to 90% assuming the refinement is not necessary across the entire domain at all times. Along with the added techniques for improving the efficiency of solving the chosen model, a technique for inserting the complex geometry of a lightning channel source was developed which can insert such a source efficiently in an algorithmic and mathematical fashion.

The model was also modified to include the effect of refraction, as the increased range of altitude in the atmosphere must account for a variation in the speed of sound. Another capability of the model that was shown was an ability to include the effects of wind as a force. The new techniques and effects introduced for the model were then validated and explored in order to establish their utility in extension of the model and the ability to convincingly simulate the process of thunder. The extension of this very general model was used for the very specific purpose of expanding upon the study of simulating the physical phenomena of thunder. However, the techniques and effects introduced do not diminish the generality of the model in any capacity and can therefore be applied to several other areas of study as well.

6.2 Future Work

When completing this work, some factors were unexplored or strayed from, due to lack of necessity, lack of resources, time constraints, and/or insurmountability. The most notable factors this author is aware of that were unexplored or not implemented will be described here as possible future work for those interested in pursuing them.

6.2.1 AMR with Conservation

When implementing AMR in this work, the fluxes at the cell boundaries were not calculated to be transferred across refinement levels; only the cell values were. Consequently, the model is not conservative across grid levels. Therefore a future addition

to the model would be to calculate these flux values in order to have the model be conservative across refinements levels. This extra effort was not a focus of this work as the model is still conservative at each refinement level, so if the shocks are always contained in the highest refinement level, the error introduced due to being non-conservative across refinement levels is tolerable. Manzanares[46] has done a considerable amount of work in accomplishing this for hyperbolic problems using a WENO scheme along with AMR. Shen *et al.*[54] employed a conservative update method in an AMR WENO scheme that generated unphysical solutions.

6.2.2 AMR Using Numerical Kernels for GPUs

When implementing AMR using SAMRAI, the calculations carried out on each patch (sometimes called the numerical kernels) are typically written in FORTRAN for reasons of easier indexing into multidimensional arrays and speed considerations. However, in the advent of GPU programming, it would be worthwhile to extend the model to use numerical kernels written in OpenCL or CUDA to speed up calculations. Preliminary tests were done with the model used in this work with a static grid version of the program written using OpenCL that saw dramatic speed gains during execution. This is due to the large amount of floating point calculations necessary in the model and the relatively bijou storage requirements. However, the solutions would break down after several time steps due to the single precision requirement of current commodity GPUs. An analysis of the situation causing the breakdown using single precision was not conducted. However, it can be seen that when small amplitude waves are used in the model, it is possible that the numerical differentiation operation can break down using single precision due to the differencing of nearly equal values.

One aspect of programming for GPUs is that when shipping a grid of values to the GPU to do calculations on, the dimensions of the grid should be powers of two in order to optimize how GPUs read and write to onboard memory. For these kernels to perform optimally in SAMRAI, it would need a new feature which would enable it to enforce patch dimensions to be powers of two. Another reason these GPU numerical kernels were not fully explored in this work was the lack of accessibility to GPUs that were capable of double precision arithmetic. At the time of writing, many GPUs on the market were capable of handling double precision. However, the cost was prohibitive, and double precision is necessary for the model, most notably for the numerical differentiation operation. With GPUs having relatively small amounts of onboard memory compared to the RAM for the CPU, the ability to ship patches of the domain to the GPU for calculation would be a valuable investigation. Antoniou *et al.*[68] gives a WENO implementation using GPUs while Schive *et al.*[69] uses GPUs in an AMR situation with a MUSCL scheme.

6.2.3 Model with Thermal Expansion

The current model begins simulation at the insertion of a density source which is effectively the time after the molecules in the lightning channel break down and have rapidly expanded to create an initial pressure shockwave. One addition to the model that would bring it closer to the physical reality of the phenomena would be to start the simulation at the insertion of the temperature source. While temperature sources were tested with this model, it appears that inserting density sources is the safest way to keep the model stable. A thorough investigation of using temperature sources and adding thermal expansion to the model would be beneficial.

6.2.4 Perfectly Matched Boundary Layer

While the absorbing boundary conditions for this work were satisfactory, the minor reflections incurred at the boundary were still undesirable. Therefore the model would benefit from a perfectly matched boundary layer (PML). Designing perfectly matched boundary layers is done on a case-by-case basis, where the model equations are modified to include a PML. Berenger[70] was the first to introduce the perfectly matched layer with some advances being made by Chevalier[71]. Johnson[72] has notes and examples of PMLs applied to the wave equation.

6.2.5 Partially Absorbing Ground Boundary

Perfectly reflecting ground boundary conditions were used in this work due to their simplicity. However, generally the ground in reality absorbs acoustics waves depending on its makeup. This addition would bring a more physically accurate result to the simulation. It may be possible to shorten the envelope of the absorbing boundary condition in way that creates the correct amount of partial absorption as well.

6.2.6 Better Spatially Varying Quantities

In the given model, only the speed of sound has been used as a spatially varying quantity. This was in part due to the absorbing boundary conditions posing a problem of propagating wrong values into the domain after every time step. However, in the atmosphere, temperature, pressure and density also vary based on altitude. A future model should account for gradient initial conditions and ambient conditions for all of these variables. Gravity would need to be included in the model to counteract the effect of higher pressures moving toward lower pressures vertically. This might possibly be achieved simply by the adding in the force of gravity as a body force in a similar manner as a wind force was introduced using the momentum equation. However, a thorough analysis of such an addition was not conducted.

6.2.7 More Efficient Source Algorithm

The algorithm for drawing the list of segments on the source grid is inefficient for drawing lightning channel geometries composed of several segments. When initially drawing the source, the algorithm iterates over each cell in each patch once for every segment in the list. For this reason, segment lengths were fixed to certain lengths so that the list of segments that describe the lightning channel was small. However, according to Few[5], tortuosity in a lightning channel occurs on all scales, and therefore a more efficient source drawing algorithm would be beneficial to mimicking this property more closely.

6.2.8 Element-Structured AMR

With block-structured adaptive mesh refinement, the grid refinement levels overlap. The coarsest grid must have cell sizes small enough to accommodate a well-posed description of the lightning channel source. Otherwise the base grid can have adverse effects on the solution in that it may create instability. With element-structured AMR, there is only one grid level and each cell is refined through a tree-type description of the grid. This allows for a wider range of refinement as well as decreased necessary storage. However, this means that each cell or element stands on its own and performing the numerical methods on one element at a time could be inefficient due to many interpolations from neighboring cells and inefficient use of processor cache since elements that are contiguous on the grid may not be contiguous in memory. The main advantage to be sought after is an implementation that does not overlap grids on different refinement levels, but the price may be a more complex AMR implemenation.

Appendix A

Sample Code for Source Fabrication

A.1 Creating the Lightning Geometry in MATLAB

```
function LightningChannel(domainW,domainH,startX,startY,...
    maxIterations,branchLevels,meanAngle,write,overwrite,N,...
    branchyness,meanBranchLength,numAvgAngles)

    %domainW,domainH are width and height of domain
    %startX,startY are x,y start of lightning channel
    %maxIterations is maximum amount of main branch segment iterations
    %branchLevels is level of recursive branching
    %meanAngle is mean of random angles
    %write==1 write to file, write==0 don't write to file
    %overwrite==1 overwrite file, overwrite==0 append to file
    %N=20 from Roy and Ribner; bias towards vertical
    %branchyness is in [0,1], lower numbers mean less branching
    %meanBranchLength is average length of branches
    %numAvgAngles is number of previous angles to use in smoothing

    x=zeros(1,2);
    y=zeros(1,2);
    i=0;
    x(1)=startX;
    y(1)=startY;
```

```
22      angles=zeros(numAvgAngles,1);
23
24      if(overwrite==1)
25          fid=fopen('lightning.txt', 'w');
26          fclose(fid);
27      end
28
29      while((y(1)>0) && (i<maxIterations))
30          theta=GetAngle(meanAngle,angles,N);
31
32          length=GetLength();
33
34          for n=numAvgAngles-1:-1:1
35              angles(n+1,1)=angles(n,1);
36          end
37
38          angles(1,1)=theta;
39
40          y(2)=y(1)-length*sind(90-theta);
41          x(2)=x(1)+length*cosd(90-theta);
42
43          if(y(2)<0)
44              y(2)=0;
45          end
46
47          if(write==1)
48              fid=fopen('lightning.txt', 'a');
49              fprintf(fid, '%f %f %f %f\n',x(1),y(1),x(2),y(2));
50              fclose(fid);
51          end
52
53          line(x, y);
54          axis([0 domainW 0 domainH]);
55          x(1)=x(2);
56          y(1)=y(2);
57          i=i+1;
58
59          if(branchLevels>0)
60              if(rand(1,1)<branchyness)
61                  branchMaxIterations=maxIterations/5;
62                  LightningChannel(domainW,domainH,x(2),y(2),...
63                      branchMaxIterations,branchLevels-1,meanAngle,...
```

```
                    write,0,N,branchyness/10,meanBranchLength/2,...
                    numAvgAngles);
            end
        end
    end
end

function angle=GetAngle(meanAngle,angles,N)
    angle=-180:180;
    P=exp((-(angle-mean(angles)+mean(angles)/N).^2)./(meanAngle^2));
    P=P./trapz(angle,P(:));
    anglei=linspace(min(angle),max(angle),10000)';
    Pi=interp1(angle,P,anglei);
    cdf=cumtrapz(anglei,Pi);
    i=[true; not(diff(cdf)==0)];
    cdf=cdf(i);
    anglei=anglei(i);
    angle=interp1(cdf,anglei,rand(1,1));
end

function length=GetLength()
    length=6;
end
```

For a two square kilometer domain, this program would be ran, for example, as: `LightningChannel(2000,2000,250,2000,700,2,16,1,1,20,0.006,8,4);`

A.2 Drawing the Lightning Source in MATLAB

```
function DrawSource(domainW,domainH,X,Y,alpha,srcAmp)

    %domainW,domainH are width and height of domain
    %X,Y are resolution of grid
    %alpha is the spread of the source
    %srcAmp is the amplitude of the source

    dx=domainW/X;
    dy=domainH/Y;
    H0=zeros(X,Y);

```

```
    fid = fopen('lightning.txt');
    S = fscanf(fid, '%f %f %f %f', [4 inf]);
    fclose(fid);

    for k=1:size(S,2)
        theta=-atan((S(4,k)-S(2,k))/(S(3,k)-S(1,k)));

        if((S(1,k)>S(3,k) && S(2,k)>S(4,k)) ||...
                (S(1,k)<=S(3,k) && S(2,k)>=S(4,k)))
            orientation=1;
        elseif((S(1,k)>=S(3,k) && S(2,k)<=S(4,k)) ||...
                (S(1,k)<S(3,k) && S(2,k)<S(4,k)))
            orientation=2;
        end

        for j=1:Y
            yWorld=dy*(j-0.5);
            for i=1:X
                xWorld=dx*(i-0.5);
                line0=tan(pi/2-theta)*(xWorld-S(1,k))+S(2,k);
                line1=tan(pi/2-theta)*(xWorld-S(3,k))+S(4,k);
                if((orientation==1 && yWorld>=line0)...
                        || (orientation==2 && yWorld<=line0))
                    fun=srcAmp*exp(-(log(2)/alpha^2)*...
                        ((xWorld-S(1,k))^2+(yWorld-S(2,k))^2));
                end
                if((orientation==1 && yWorld<=line1)...
                        || (orientation==2 && yWorld>=line1))
                    fun=srcAmp*exp(-(log(2)/alpha^2)*...
                        ((xWorld-S(3,k))^2+(yWorld-S(4,k))^2));
                end
                if((orientation==1 && yWorld<line0 && yWorld>line1)...
                        || (orientation==2 && yWorld>line0 &&...
                        yWorld<line1))
                    fun=srcAmp*exp(-(log(2)/alpha^2)*...
                        (sin(theta)*(xWorld-S(1,k))...
                        +cos(theta)*(yWorld-S(2,k)))^2);
                end
                H0(i,j)=H0(i,j)+fun*(1-(H0(i,j)/srcAmp));
            end
        end
    end
```

```
54      imagesc(rot90(H0));
55  end
```

This program would be ran, for example, as:
`DrawSource(1000,1000,2000,2000,6,1);`

Appendix B

Sample Code for Hybrid Model

In this appendix, sample MATLAB code is given for the hybrid model based on the third-order Runge-Kutta and fifth-order WENO schemes. The model is implemented including molecular relaxation due to nitrogen and using a global Lax-Friedrichs flux-splitting on a static mesh. Only point-source sources are implemented as well as global reflecting or absorbing boundary conditions. MATLAB is not the correct tool to use for running the program on anything but small problems. The focus of this code is to allow it to be understood in a simple fashion and easily ported to another language such as C. This code is optimized to use a minimal amount of memory as a tradeoff between more computation due to recalculating certain values rather than storing them. It is not optimized for running in MATLAB as it is not properly vectorized and has been written mostly using loops. When profiling, it can be seen that the code will spend most of its time in the `rhs()` function. In the C language, the program written in a form close to the given sample code can be parallelized easily by a straightforward use of the OpenMP library on most of the outermost loops for example. No error checking is involved for the sake of brevity and it is set to only plot the domain every 25 time steps for better performance.

```
function hybrid(timeSteps,X,Y,srcPosX,srcPosY,...
    srcAmp,srcAlpha,maxFreq,ppw,cfl,bc,rc)

    %timeSteps is number of time steps to run
    %X,Y are number of cells of domain
    %srcPosX,srcPosY are the center location of source
    %srcAmp is density amplitude of source
    %srcAlpha is spread of point source
    %maxFreq is maximum frequency to account for
    %ppw is points per width of maxFreq wavelength
    %cfl condtion
```

```
12  %bc, 1 for reflecting boundaries, 2 for absorbing
13  %rc is hybrid threshold for subscheme switching
14
15  global w_n w p T H1 tag;
16
17  alpha=360;
18  c=343;
19  mygamma=1.402;
20  p0=101325;
21  p_ref=p0;
22  RH=20;
23  T0=293.16;
24  T_ref=T0;
25  rho0=1.21;
26  R=287.06;
27  c_v=720.4;
28  c_p=1010;
29  T_star_n=3352;
30  mu=1.846e-5;
31  muB=0.6*mu;
32  kappa=0.02624;
33
34  w=zeros(X,Y,5);
35  H1=zeros(X,Y);
36  p=zeros(X,Y);
37  T=zeros(X,Y);
38  tag=zeros(X,Y);
39
40  lambda=c/maxFreq;
41  dx=lambda/ppw;
42  dy=lambda/ppw;
43  dt=cfl*(dx/c);
44
45  zeta=10.79586*(1-(273.16/T0))-5.02808*...
46      log10(T0/273.16)+(1.50474e-4)*...
47      (1-10^(-8.29692*((T0/273.16)-1)))-...
48      (4.2873e-4)*(1-10^(-4.76955*...
49      ((273.16/T0)-1)))-2.2195983;
50  p_vp=p_ref*10^zeta;
51  h=(RH*p_vp)/p0;
52  eta=(4.17)*(((T_ref/T0)^(1/3))-1);
53  f_n=(p0/p_ref)*(((T_ref/T0)^(1/2))*...
```

```
54          (9+280*h*exp(-eta)));
55      tau_n=1/(2*pi*f_n);
56
57      if dt>((tau_n)/4)
58          dt=(tau_n)/4;
59      end
60
61      p(:,:)=p0;
62      T(:,:)=T0;
63      w(:,:,1)=rho0;
64      w(:,:,2)=0;
65      w(:,:,3)=0;
66      w(:,:,4)=0;
67      w(:,:,5)=rho0*T0;
68      w_n=w;
69
70      updateState(p0,T0,rho0,R,c_v,c_p,mygamma,c);
71
72      addSource(X,Y,srcPosX,srcPosY,...
73          srcAmp,srcAlpha);
74
75      rungeKutta(X,Y,dx,dy,muB,mu,...
76              tau_n,T_star_n,R,kappa,...
77              alpha,dt,bc,rc,p0,T0,rho0,mygamma,c,c_v,c_p);
78      H1(:,:)=0;
79
80      for step=2:timeSteps
81          rungeKutta(X,Y,dx,dy,muB,mu,...
82              tau_n,T_star_n,R,kappa,...
83              alpha,dt,bc,rc,p0,T0,rho0,mygamma,c,c_v,c_p);
84
85          if mod(step,25)==0
86              %subplot(1,2,1);
87              surf(p-p0);
88              axis([1 Y 1 X -srcAmp/10 srcAmp/10]);
89              xlabel('y');
90              ylabel('x');
91              title('p');
92
93              %subplot(1,2,2);
94              %image(tag.*256);
95              %xlabel('y');
```

```
96              %ylabel('x');
97              %title('tags');
98              drawnow;
99          end
100     end
101 end
102
103 function rungeKutta(X,Y,dx,dy,muB,mu,...
104     tau_n,T_star_n,R,kappa,alpha,...
105     dt,bc,rc,p0,T0,rho0,mygamma,c,c_v,c_p)
106     global w w_n;
107
108     K=rhs(X,Y,dx,dy,muB,mu,...
109     tau_n,T_star_n,R,kappa,alpha,rc);
110     w_n=w+dt*K;
111     updateState(p0,T0,rho0,R,c_v,c_p,mygamma,c);
112     boundaryConditions(bc,X,Y,p0,T0,rho0);
113
114     K=rhs(X,Y,dx,dy,muB,mu,...
115     tau_n,T_star_n,R,kappa,alpha,rc);
116     w_n=(3/4)*w+(1/4)*w_n+(1/4)*dt*K;
117     updateState(p0,T0,rho0,R,c_v,c_p,mygamma,c);
118     boundaryConditions(bc,X,Y,p0,T0,rho0);
119
120     K=rhs(X,Y,dx,dy,muB,mu,...
121     tau_n,T_star_n,R,kappa,alpha,rc);
122     w_n=(1/3)*w+(2/3)*w_n+(2/3)*dt*K;
123     updateState(p0,T0,rho0,R,c_v,c_p,mygamma,c);
124     boundaryConditions(bc,X,Y,p0,T0,rho0);
125
126     w=w_n;
127 end
128
129 function K=rhs(X,Y,dx,dy,muB,mu,...
130     tau_n,T_star_n,R,kappa,alpha,rc)
131     global w_n H1 p T tag;
132
133     K=zeros(X,Y,5);
134     F=zeros(7,5);
135     G=zeros(7,5);
136     H=zeros(1,5);
137     tag(:,:)=0;
```

```
138     xi=1;
139     ep=0.9*rc*xi*xi/(1-0.9*rc*rc);
140     a_1=0.79926643;
141     a_2=-0.18941314;
142     a_3=0.02651995;
143
144     for i=4:X-3
145         for j=4:Y-3
146             ix=4;
147             for q=-3:3
148                F(ix+q,1)=w_n(i+q,j,1)*...
149                    (w_n(i+q,j,2)/w_n(i+q,j,1));
150                G(ix+q,1)=w_n(i,j+q,1)*...
151                    (w_n(i,j+q,3)/w_n(i,j+q,1));
152
153                F(ix+q,2)=w_n(i+q,j,1)*...
154                    ((w_n(i+q,j,2)/w_n(i+q,j,1))^2)+...
155                    p(i+q,j);
156                G(ix+q,2)=w_n(i,j+q,1)*...
157                    (w_n(i,j+q,3)/w_n(i,j+q,1))*...
158                    (w_n(i,j+q,2)/w_n(i,j+q,1));
159
160                F(ix+q,3)=w_n(i+q,j,1)*...
161                    (w_n(i+q,j,2)/w_n(i+q,j,1))*...
162                    (w_n(i+q,j,3)/w_n(i+q,j,1));
163                G(ix+q,3)=w_n(i,j+q,1)*...
164                    ((w_n(i,j+q,3)/w_n(i,j+q,1))^2)+...
165                    p(i,j+q);
166
167                F(ix+q,4)=w_n(i+q,j,1)*...
168                    (w_n(i+q,j,2)/w_n(i+q,j,1))*...
169                    (w_n(i+q,j,4)/w_n(i+q,j,1));
170                G(ix+q,4)=w_n(i,j+q,1)*...
171                    (w_n(i,j+q,3)/w_n(i,j+q,1))*...
172                    (w_n(i,j+q,4)/w_n(i,j+q,1));
173
174                F(ix+q,5)=w_n(i+q,j,1)*...
175                    (w_n(i+q,j,2)/w_n(i+q,j,1))*...
176                    (w_n(i+q,j,5)/w_n(i+q,j,1));
177                G(ix+q,5)=w_n(i,j+q,1)*...
178                    (w_n(i,j+q,3)/w_n(i,j+q,1))*...
179                    (w_n(i,j+q,5)/w_n(i,j+q,1));
```

```
180             end
181
182             H(1)=H1(i,j);
183
184             u_dprime_x=((w_n(i-1,j,2)/w_n(i-1,j,1))-...
185                 2*(w_n(i,j,2)/w_n(i,j,1))+...
186                 (w_n(i+1,j,2)/w_n(i+1,j,1)))/(dx^2);
187             u_dprime_y=((w_n(i,j-1,2)/w_n(i,j-1,1))-...
188                 2*(w_n(i,j,2)/w_n(i,j,1))+...
189                 (w_n(i,j+1,2)/w_n(i,j+1,1)))/(dy^2);
190             v_dprime_xy=...
191                 ((w_n(i+1,j+1,3)/w_n(i+1,j+1,1))-...
192                 (w_n(i+1,j-1,3)/w_n(i+1,j-1,1))-...
193                 (w_n(i-1,j+1,3)/w_n(i-1,j+1,1))+...
194                 (w_n(i-1,j-1,3)/w_n(i-1,j-1,1)))/...
195                 (4*dx*dy);
196             phi_xx_prime_x=(4.0/3.0)*...
197                 u_dprime_x-(2.0/3.0)*v_dprime_xy;
198             phi_xy_prime_y=u_dprime_y+v_dprime_xy;
199
200             H(2)=muB*(u_dprime_x+v_dprime_xy)+...
201                 mu*(phi_xx_prime_x+phi_xy_prime_y);
202
203             v_dprime_y=((w_n(i,j-1,3)/w_n(i,j-1,1))-...
204                 2*(w_n(i,j,3)/w_n(i,j,1))+...
205                 (w_n(i,j+1,3)/w_n(i,j+1,1)))/(dy^2);
206             v_dprime_x=((w_n(i-1,j,3)/w_n(i-1,j,1))-...
207                 2*(w_n(i,j,3)/w_n(i,j,1))+...
208                 (w_n(i+1,j,3)/w_n(i+1,j,1)))/(dx^2);
209             u_dprime_yx=...
210                 ((w_n(i+1,j+1,2)/w_n(i+1,j+1,1))-...
211                 (w_n(i+1,j-1,2)/w_n(i+1,j-1,1))-...
212                 (w_n(i-1,j+1,2)/w_n(i-1,j+1,1))+...
213                 (w_n(i-1,j-1,2)/w_n(i-1,j-1,1)))/...
214                 (4*dx*dy);
215             phi_yx_prime_x=u_dprime_yx+v_dprime_x;
216             phi_yy_prime_y=(4.0/3.0)*...
217                 v_dprime_y-(2.0/3.0)*u_dprime_yx;
218
219             H(3)=muB*(v_dprime_y+u_dprime_yx)+...
220                 mu*(phi_yx_prime_x+phi_yy_prime_y);
221
```

```
222             u_prime_x=((w_n(i+1,j,2)/w_n(i+1,j,1))-...
223                 (w_n(i-1,j,2)/w_n(i-1,j,1)))/(2*dx);
224             u_prime_y=((w_n(i,j+1,2)/w_n(i,j+1,1))-...
225                 (w_n(i,j-1,2)/w_n(i,j-1,1)))/(2*dy);
226             v_prime_y=((w_n(i,j+1,3)/w_n(i,j+1,1))-...
227                 (w_n(i,j-1,3)/w_n(i,j-1,1)))/(2*dy);
228             v_prime_x=((w_n(i+1,j,3)/w_n(i+1,j,1))-...
229                 (w_n(i-1,j,3)/w_n(i-1,j,1)))/(2*dx);
230             t_prime_x=(T(i+1,j)-T(i-1,j))/(2*dx);
231             t_prime_y=(T(i,j+1)-T(i,j-1))/(2*dy);
232             t_dprime_x=(T(i-1,j)-2*T(i,j)+...
233                 T(i+1,j))/(dx^2);
234             t_dprime_y=(T(i,j-1)-2*T(i,j)+...
235                 T(i,j+1))/(dy^2);
236             T_n=w_n(i,j,5)/w_n(i,j,1);
237             Mn=(T(i,j)-T_n)/tau_n;
238             tmp1=T_star_n/(T_n);
239             c_vn=0.78*R*(tmp1^2)*exp(-tmp1);
240             A_n=((T(i,j)/T_n)-1)*c_vn;
241             phi_xx=(4.0/3.0)*u_prime_x-(2.0/3.0)*v_prime_y;
242             phi_xy_yx=u_prime_y+v_prime_x;
243             phi_yy=(4.0/3.0)*v_prime_y-(2.0/3.0)*u_prime_x;
244             sigma=(muB/T(i,j))*((u_prime_x+v_prime_y)^2)+...
245                 (mu/(2*T(i,j)))*...
246                 (phi_xx^2+2*(phi_xy_yx^2)+phi_yy^2)+...
247                 (kappa/(T(i,j)^2))*...
248                 (t_prime_x^2+t_prime_y^2)+...
249                 (w_n(i,j,1)/T(i,j))*...
250                 (A_n*Mn);
251             H(4)=sigma-...
252                 ((w_n(i,j,1)/T_n)*c_vn*Mn)+...
253                 (-((kappa*(t_prime_x^2))/(T(i,j)^2))+...
254                 ((kappa*t_dprime_x)/T(i,j))-...
255                 ((kappa*(t_prime_y^2))/(T(i,j)^2))+...
256                 ((kappa*t_dprime_y)/T(i,j)));
257
258             H(5)=(w_n(i,j,1)/tau_n)*(T(i,j)-T_n);
259
260             for q=-1:1
261                 tmp1=w_n(i+q,j,1)-w_n(i-1+q,j,1);
262                 tmp2=w_n(i+1+q,j,1)-w_n(i+q,j,1);
263                 rx=(2*abs(tmp1*tmp2)+ep)/(tmp1^2+tmp2^2+ep);
```

```
264                 if (rx<0.9999)
265                     tag(i,j)=1;
266                 end
267
268                 tmp1=w_n(i,j+q,1)-w_n(i,j-1+q,1);
269                 tmp2=w_n(i,j+1+q,1)-w_n(i,j+q,1);
270                 ry=(2*abs(tmp1*tmp2)+ep)/(tmp1^2+tmp2^2+ep);
271                 if (ry<0.9999)
272                     tag(i,j)=1;
273                 end
274             end
275
276             if tag(i,j)==0
277                 ix=4;
278                 for k=1:5
279                     K(i,j,k)=...
280                         (-a_3*F(ix-3,k)-a_2*F(ix-2,k)-...
281                         a_1*F(ix-1,k)+a_1*F(ix+1,k)+...
282                         a_2*F(ix+2,k)+a_3*F(ix+3,k))/-dx...
283                         +(-a_3*G(ix-3,k)-a_2*G(ix-2,k)-...
284                         a_1*G(ix-1,k)+a_1*G(ix+1,k)+...
285                         a_2*G(ix+2,k)+a_3*G(ix+3,k))/-dy...
286                         +H(k);
287                 end
288             else
289                 for k=1:5
290                     ii=i-1;
291                     ix=4-1;
292                     fgm2=0.5*(F(ix-2,k)+alpha*w_n(ii-2,j,k));
293                     fgm1=0.5*(F(ix-1,k)+alpha*w_n(ii-1,j,k));
294                     fg0 =0.5*(F(ix  ,k)+alpha*w_n(ii  ,j,k));
295                     fgp1=0.5*(F(ix+1,k)+alpha*w_n(ii+1,j,k));
296                     fgp2=0.5*(F(ix+2,k)+alpha*w_n(ii+2,j,k));
297                     pf1=wenoP(fgm2,fgm1,fg0,fgp1,fgp2);
298
299                     ii=ii+1;
300                     ix=ix+1;
301                     fgm2=0.5*(F(ix-2,k)-alpha*w_n(ii-2,j,k));
302                     fgm1=0.5*(F(ix-1,k)-alpha*w_n(ii-1,j,k));
303                     fg0 =0.5*(F(ix  ,k)-alpha*w_n(ii  ,j,k));
304                     fgp1=0.5*(F(ix+1,k)-alpha*w_n(ii+1,j,k));
305                     fgp2=0.5*(F(ix+2,k)-alpha*w_n(ii+2,j,k));
```

```
306                 mf1=wenoM(fgm2,fgm1,fg0,fgp1,fgp2);
307
308                 fgm2=0.5*(F(ix-2,k)+alpha*w_n(ii-2,j,k));
309                 fgm1=0.5*(F(ix-1,k)+alpha*w_n(ii-1,j,k));
310                 fg0 =0.5*(F(ix  ,k)+alpha*w_n(ii  ,j,k));
311                 fgp1=0.5*(F(ix+1,k)+alpha*w_n(ii+1,j,k));
312                 fgp2=0.5*(F(ix+2,k)+alpha*w_n(ii+2,j,k));
313                 pf2=wenoP(fgm2,fgm1,fg0,fgp1,fgp2);
314
315                 ii=ii+1;
316                 ix=ix+1;
317                 fgm2=0.5*(F(ix-2,k)-alpha*w_n(ii-2,j,k));
318                 fgm1=0.5*(F(ix-1,k)-alpha*w_n(ii-1,j,k));
319                 fg0 =0.5*(F(ix  ,k)-alpha*w_n(ii  ,j,k));
320                 fgp1=0.5*(F(ix+1,k)-alpha*w_n(ii+1,j,k));
321                 fgp2=0.5*(F(ix+2,k)-alpha*w_n(ii+2,j,k));
322                 mf2=wenoM(fgm2,fgm1,fg0,fgp1,fgp2);
323
324                 jj=j-1;
325                 ix=4-1;
326                 fgm2=0.5*(G(ix-2,k)+alpha*w_n(i,jj-2,k));
327                 fgm1=0.5*(G(ix-1,k)+alpha*w_n(i,jj-1,k));
328                 fg0 =0.5*(G(ix  ,k)+alpha*w_n(i,jj  ,k));
329                 fgp1=0.5*(G(ix+1,k)+alpha*w_n(i,jj+1,k));
330                 fgp2=0.5*(G(ix+2,k)+alpha*w_n(i,jj+2,k));
331                 pg1=wenoP(fgm2,fgm1,fg0,fgp1,fgp2);
332
333                 jj=jj+1;
334                 ix=ix+1;
335                 fgm2=0.5*(G(ix-2,k)-alpha*w_n(i,jj-2,k));
336                 fgm1=0.5*(G(ix-1,k)-alpha*w_n(i,jj-1,k));
337                 fg0 =0.5*(G(ix  ,k)-alpha*w_n(i,jj  ,k));
338                 fgp1=0.5*(G(ix+1,k)-alpha*w_n(i,jj+1,k));
339                 fgp2=0.5*(G(ix+2,k)-alpha*w_n(i,jj+2,k));
340                 mg1=wenoM(fgm2,fgm1,fg0,fgp1,fgp2);
341
342                 fgm2=0.5*(G(ix-2,k)+alpha*w_n(i,jj-2,k));
343                 fgm1=0.5*(G(ix-1,k)+alpha*w_n(i,jj-1,k));
344                 fg0 =0.5*(G(ix  ,k)+alpha*w_n(i,jj  ,k));
345                 fgp1=0.5*(G(ix+1,k)+alpha*w_n(i,jj+1,k));
346                 fgp2=0.5*(G(ix+2,k)+alpha*w_n(i,jj+2,k));
347                 pg2=wenoP(fgm2,fgm1,fg0,fgp1,fgp2);
```

```
348
349                     jj=jj+1;
350                     ix=ix+1;
351                     fgm2=0.5*(G(ix-2,k)-alpha*w_n(i,jj-2,k));
352                     fgm1=0.5*(G(ix-1,k)-alpha*w_n(i,jj-1,k));
353                     fg0 =0.5*(G(ix  ,k)-alpha*w_n(i,jj  ,k));
354                     fgp1=0.5*(G(ix+1,k)-alpha*w_n(i,jj+1,k));
355                     fgp2=0.5*(G(ix+2,k)-alpha*w_n(i,jj+2,k));
356                     mg2=wenoM(fgm2,fgm1,fg0,fgp1,fgp2);
357
358                     K(i,j,k)=(pf1+mf1-pf2-mf2)/dx+...
359                         (pg1+mg1-pg2-mg2)/dy+H(k);
360                 end
361             end
362         end
363     end
364 end
365
366 function fm=wenoM(fgm2,fgm1,fg0,fgp1,fgp2)
367     ep=10e-6;
368
369     beta0=(13/12)*(fg0-2*fgp1+fgp2)^2+...
370         (1/4)*(3*fg0-4*fgp1+fgp2)^2;
371     beta1=(13/12)*(fgm1-2*fg0+fgp1)^2+...
372         (1/4)*(fgm1-fgp1)^2;
373     beta2=(13/12)*(fgm2-2*fgm1+fg0)^2+...
374         (1/4)*(fgm2-4*fgm1+3*fg0)^2;
375
376     zeta0=0.1/((ep+beta0)^2);
377     zeta1=0.6/((ep+beta1)^2);
378     zeta2=0.3/((ep+beta2)^2);
379
380     sigma=zeta0+zeta1+zeta2;
381
382     omega0=zeta0/sigma;
383     omega1=zeta1/sigma;
384     omega2=zeta2/sigma;
385
386     f0=(1/6)*(11*fg0-7*fgp1+2*fgp2);
387     f1=(1/6)*(2*fgm1+5*fg0-fgp1);
388     f2=(1/6)*(-fgm2+5*fgm1+2*fg0);
389
```

```
390     fm=omega0*f0+omega1*f1+omega2*f2;
391 end
392
393 function fp=wenoP(fgm2,fgm1,fg0,fgp1,fgp2)
394     ep=10e-6;
395
396     beta0=(13/12)*(fg0-2*fgp1+fgp2)^2+...
397         (1/4)*(3*fg0-4*fgp1+fgp2)^2;
398     beta1=(13/12)*(fgm1-2*fg0+fgp1)^2+...
399         (1/4)*(fgm1-fgp1)^2;
400     beta2=(13/12)*(fgm2-2*fgm1+fg0)^2+...
401         (1/4)*(fgm2-4*fgm1+3*fg0)^2;
402
403     zeta0=0.3/((ep+beta0)^2);
404     zeta1=0.6/((ep+beta1)^2);
405     zeta2=0.1/((ep+beta2)^2);
406
407     sigma=zeta0+zeta1+zeta2;
408
409     omega0=zeta0/sigma;
410     omega1=zeta1/sigma;
411     omega2=zeta2/sigma;
412
413     f0=(1/6)*(2*fg0+5*fgp1-fgp2);
414     f1=(1/6)*(-fgm1+5*fg0+2*fgp1);
415     f2=(1/6)*(2*fgm2-7*fgm1+11*fg0);
416
417     fp=omega0*f0+omega1*f1+omega2*f2;
418 end
419
420 function updateState(p0,T0,rho0,R,c_v,c_p,mygamma,c)
421     global w_n p T;
422
423     p=(c^2).*((w_n(:,:,1)-rho0)+((mygamma-1)...
424         ./(2.*rho0)).*((w_n(:,:,1)-rho0).^2)...
425         +(w_n(:,:,1)./c_p)...
426         .*(w_n(:,:,4)./w_n(:,:,1)))+p0;
427     T=T0.*exp((w_n(:,:,4)./w_n(:,:,1)...
428         -R.*log(rho0./w_n(:,:,1))))./c_v);
429 end
430
431 function addSource(X,Y,srcPosX,srcPosY,...
```

```
    srcAmp,srcAlpha)
    global H1;

    for i=1:X
        for j=1:Y
            H1(i,j)=srcAmp*exp(-log(2)/((srcAlpha)^2)...
               *(((i-srcPosX)^2)+((j-srcPosY)^2)));
        end
    end
end

function boundaryConditions(bc,X,Y,p0,T0,rho0)
    if bc==1
        bcReflect(X,Y);
    elseif bc==2
        bcAbsorb(X,Y,p0,T0,rho0);
    end
end

function bcReflect(X,Y)
    global w_n;

    w_n(1,:,:)=w_n(4,:,:);
    w_n(2,:,:)=w_n(4,:,:);
    w_n(3,:,:)=w_n(4,:,:);
    w_n(3,:,2)=-w_n(4,:,2);

    w_n(:,1,:)=w_n(:,4,:);
    w_n(:,2,:)=w_n(:,4,:);
    w_n(:,3,:)=w_n(:,4,:);
    w_n(:,3,3)=-w_n(:,4,3);

    w_n(X,:,:)=w_n(X-3,:,:);
    w_n(X-1,:,:)=w_n(X-3,:,:);
    w_n(X-2,:,:)=w_n(X-3,:,:);
    w_n(X-2,:,2)=-w_n(X-3,:,2);

    w_n(:,Y,:)=w_n(:,Y-3,:);
    w_n(:,Y-1,:)=w_n(:,Y-3,:);
    w_n(:,Y-2,:)=w_n(:,Y-3,:);
    w_n(:,Y-2,3)=-w_n(:,Y-3,3);
end
```

```
474
475 function bcAbsorb(X,Y,p0,T0,rho0)
476     global w_n p T;
477
478     numpoints=8;
479     x_s=1; y_s=1;
480     envAlpha=4;
481
482     for i=1:numpoints
483         tmp=(-exp(-log(2)/(envAlpha^2)*(i-1-x_s)^2)+1);
484         w_n(i,:,1)=(w_n(i,:,1)-rho0)*tmp+rho0;
485         w_n(i,:,2)=w_n(i,:,2)*tmp;
486         w_n(i,:,3)=w_n(i,:,3)*tmp;
487         w_n(i,:,4)=w_n(i,:,4)*tmp;
488         w_n(i,:,5)=(w_n(i,:,5)-T0*rho0)*tmp+T0*rho0;
489         p(i,:)=(p(i,:)-p0)*tmp+p0;
490         T(i,:)=(T(i,:)-T0)*tmp+T0;
491     end
492
493     for j=1:numpoints
494         tmp=(-exp(-log(2)/(envAlpha^2)*(j-1-y_s)^2)+1);
495         w_n(:,j,1)=(w_n(:,j,1)-rho0)*tmp+rho0;
496         w_n(:,j,2)=w_n(:,j,2)*tmp;
497         w_n(:,j,3)=w_n(:,j,3)*tmp;
498         w_n(:,j,4)=w_n(:,j,4)*tmp;
499         w_n(:,j,5)=(w_n(:,j,5)-T0*rho0)*tmp+T0*rho0;
500         p(:,j)=(p(:,j)-p0)*tmp+p0;
501         T(:,j)=(T(:,j)-T0)*tmp+T0;
502     end
503
504     for i=X:-1:X-numpoints+1
505         tmp=(-exp(-log(2)/(envAlpha^2)*(X-i-x_s)^2)+1);
506         w_n(i,:,1)=(w_n(i,:,1)-rho0)*tmp+rho0;
507         w_n(i,:,2)=w_n(i,:,2)*tmp;
508         w_n(i,:,3)=w_n(i,:,3)*tmp;
509         w_n(i,:,4)=w_n(i,:,4)*tmp;
510         w_n(i,:,5)=(w_n(i,:,5)-T0*rho0)*tmp+T0*rho0;
511         p(i,:)=(p(i,:)-p0)*tmp+p0;
512         T(i,:)=(T(i,:)-T0)*tmp+T0;
513     end
514
515     for j=Y:-1:Y-numpoints+1
```

```
516         tmp=(-exp(-log(2)/(envAlpha^2)*(Y-j-y_s)^2)+1);
517         w_n(:,j,1)=(w_n(:,j,1)-rho0)*tmp+rho0;
518         w_n(:,j,2)=w_n(:,j,2)*tmp;
519         w_n(:,j,3)=w_n(:,j,3)*tmp;
520         w_n(:,j,4)=w_n(:,j,4)*tmp;
521         w_n(:,j,5)=(w_n(:,j,5)-T0*rho0)*tmp+T0*rho0;
522         p(:,j)=(p(:,j)-p0)*tmp+p0;
523         T(:,j)=(T(:,j)-T0)*tmp+T0;
524     end
525 end
```

This program would be ran, for example, as:
`hybrid(400,51,51,26,26,1e5,1,1000,10,0.1,1,0.001);`

Bibliography

[1] W. M. Wright and N. W. Medendorp, "Acoustic Radiation from a Finite Line Source with N-Wave Excitation," *The Journal of the Acoustical Society of America*, vol. 43, no. 5, pp. 966–971, 1968.

[2] M. N. Plooster, "Numerical Model of the Return Stroke of the Lightning Discharge," *Physics of Fluids*, vol. 14, no. 10, pp. 2124–2133, 1971.

[3] H. E. Bass, "The propagation of thunder through the atmosphere," *The Journal of the Acoustical Society of America*, vol. 67, no. 6, pp. 1959–1966, 1980.

[4] M. J. Crocker and R. Raspet, *Shock Waves, Blast Waves, and Sonic Booms*, ch. 27, pp. 293–303. Wiley-Interscience, 1998.

[5] A. A. Few, *CRC Handbook of Atmospherics*, vol. 2 of *Acoustic Radiations from Lightning*. Boca Raton, FL: CRC Press, 1982.

[6] H. S. Ribner and D. Roy, "Acoustics of thunder: A quasilinear model for tortuous lightning," *The Journal of the Acoustical Society of America*, vol. 72, no. 6, pp. 1911–1925, 1982.

[7] V. W. Sparrow and R. Raspet, "A numerical method for general finite amplitude wave propagation in two dimensions and its application to spark pulses," *The Journal of the Acoustical Society of America*, vol. 90, no. 5, pp. 2683–2691, 1991.

[8] M. Wochner, *Numerical Simulation of Multi-Dimensional Acoustic Propagation in Air Including the Effects of Molecular Relaxation*. PhD thesis, Pennsylvania State University, 2006.

[9] A. S. Glassner, "The Digital Ceraunoscope: Synthetic Thunder and Lightning," tech. rep., Microsoft Research, 1999.

[10] K. Matsuyama, T. Fujimoto, and N. Chiba, "Generating Spark Discharge Sound for Interactive Animation using WM-wave," *Journal of the Society for Art and Science*, vol. 7, no. 4, pp. 145–154, 2007.

[11] K. Matsuyama, T. Fujimoto, and N. Chiba, "Real-Time Sound Generation of Spark Discharge," in *The 15th Pacific Conference on Computer Graphics and Applications*, pp. 423–426, 2007.

[12] M. F. Hamilton and M. J. Crocker, *Introduction*, ch. 1, pp. 3–19. Wiley-Interscience, 1998.

[13] M. F. Hamilton and M. J. Crocker, *Some Model Equations of Nonlinear Acoustics*, ch. 16, pp. 181–186. Wiley-Interscience, 1998.

[14] T. Leissing, "Nonlinear outdoor sound propagation," Master's thesis, Chalmers University of Technology, Göteborg, Sweden, 2006.

[15] A. D. Pierce, *Acoustics: An Introduction to Its Physical Principles and Applications*. Acoustical Society of America, 1989.

[16] D. Roy, "A Monte Carlo Model of Tortuous Lightning and the Generation of Thunder." University of Toronto, Institute for Aerospace Studies, 1981.

[17] A. A. Few, "Power Spectrum of Thunder," *Journal of Geophysical Research*, vol. 74, no. 28, pp. 6926–6934, 1969.

[18] C. R. Holmes, M. Brook, P. Krehbiel, and R. McCrory, "On the Power Spectrum and Mechanism of Thunder," *Journal of Geophysical Research*, vol. 76, no. 9, pp. 2106–2115, 1971.

[19] V. A. Rakov and M. A. Uman, *Thunder*, ch. 11, pp. 374–393. Cambridge University Press, 2007.

[20] D. T. Blackstock and M. J. Crocker, *Nonlinear Acoustics and Cavitation*, ch. 15, pp. 177–179. Wiley-Interscience, 1998.

[21] D. T. Blackstock, *History of Nonlinear Acoustics: 1750s-1930s*, ch. 1, pp. 1–23. Academic Press, 1998.

[22] R. T. Beyer, *The Parameter B/A*, ch. 2, pp. 25–39. Academic Press, 1998.

[23] D. T. Blackstock, M. F. Hamilton, and A. D. Pierce, *Progressive Waves in Lossless and Lossy Fluids*, ch. 4, pp. 65–150. Academic Press, 1998.

[24] S. Li, "WENO Schemes for Cylindrical and Spherical Geometry," tech. rep., Theoretical Division, Los Alamos Laboratory, 2003.

[25] H. E. Bass, L. C. Sutherland, A. J. Zuckerwar, D. T. Blackstock, and D. M. Hester, "Atmospheric absorption of sound: Further developments," *The Journal of the Acoustical Society of America*, vol. 97, no. 1, pp. 680–683, 1995.

[26] Equation 2.15 is Equation 2.2 in Wochner[8].

[27] S. K. Godunov, "Finite difference method for the numerical computation of discontinuous solutions of the equations of fluid dynamics," *Math Sbornik*, vol. 47, pp. 271–306, 1959.

[28] P. Lax and B. Wendroff, "Systems of conservations laws," *Communications on Pure and Applied Mathematics*, vol. 13, pp. 217–237, 1960.

[29] R. MacCormack, "The effect of viscosity in hypervelocity impact cratering," *AIAA Hypervelocity Impact Conference*, pp. 69–354, 1969.

[30] C.-W. Shu, "High Order Finite Difference and Finite Volume WENO Schemes and Discontinuous Galerkin Methods for CFD," tech. rep., NASA Langley Research Center, 2001.

[31] J. Cheng and C.-W. Shu, "High Order Schemes for CFD: A Review," *Chinese Journal of Computational Physics*, 2009.

[32] S. Gottlieb and C.-W. Shu, "Total Variation Diminishing Runge-Kutta Schemes," *Mathematics of Computation*, vol. 67, no. 221, pp. 73–85, 1998.

[33] C.-W. Shu, "Essentially Non-Oscillatory and Weighted Essentially Non-Oscillatory Schemes for Hyperbolic Conservation Laws," tech. rep., Insitute for Computer Applications in Science and Engineering, 1997.

[34] C.-W. Shu and G.-S. Jiang, "Efficient Implementation of Weighted ENO Schemes," *Journal of Computational Physics*, vol. 126, pp. 202–228, 1996.

[35] C.-W. Shu and S. Osher, "Efficient implementation of essentially non-oscillatory shock-capturing schemes ii," *Journal of Computational Physics*, vol. 83, pp. 32–78, 1989.

[36] S. Osher and S. Chakravarthy, "Upwind schemes and boundary conditions with applications to euler equations in general geometries," *Journal of Computational Physics*, vol. 50, pp. 447–481, 1983.

[37] B. Van Leer, "Towards the ultimate conservative difference scheme v. a second order sequel to godunovs method," *Journal of Computational Physics*, vol. 32, pp. 101–136, 1979.

[38] R. Donat and A. Marquina, "Capturing Shock Reflections: an Improved Flux Formula," *Journal of Computational Physics*, vol. 125, no. 1, pp. 42–58, 1996.

[39] C.-W. Shu and S. Osher, "Efficient implementation of essentially non-oscillatory shock-capturing schemes," *Journal of Computational Physics*, vol. 77, pp. 439–471, 1988.

[40] C. Tam and J. Webb, "Dispersion-Relation-Preserving Finite Difference Schemes for Computational Acoustics," *Journal of Computational Physics*, vol. 107, pp. 262–281, Jan. 1993.

[41] Y.-X. Ren, M. Liu, and H. Zhang, "A characteristic-wise hybrid compact-WENO scheme for solving hyperbolic conservation laws," *Journal of Computational Physics*, vol. 192, no. 2, pp. 365–386, 2003.

[42] D. Kim and J. H. Kwon, "A high-order accurate hybrid scheme using a central flux scheme and a WENO scheme for compressible flowfield analysis," *Journal of Computational Physics*, vol. 210, no. 2, pp. 554–583, 2005.

[43] Y. Shen and G. Yang, "Hybrid finite compact-WENO schemes for shock calculation," *International Journal for Numerical Methods in Fluids*, vol. 53, no. 4, pp. 531–560, 2007.

[44] A. H. Abdalla, Y. H. Zahran, and A. Kaltavev, "A Hybrid WENO Scheme for Conservation Laws," *Applied Mathematical Sciences*, vol. 4, no. 67, pp. 3327–3344, 2010.

[45] S. Pirozzoli, "Conservative hybrid compact-WENO schemes for shock-turbulence interaction," *Journal of Computational Physics*, vol. 178, no. 1, pp. 81–117, 2002.

[46] A. Manzanares, *Adaptive mesh refinement techniques for high order shock capturing schemes for hyperbolic systems of conservation laws*. PhD thesis, Universitat de València, Jan. 2010.

[47] T. Kim and M. C. Lin, "Physically Based Animation and Rendering of Lightning," in *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference*, 2004.

[48] T. Kim and M. C. Lin, "Fast animation of lightning using an adaptive mesh," *IEEE Transactions on Visualization and Computer Graphics*, pp. 390–402, 2007.

[49] K. Matsuyama, T. Fujimoto, and N. Chiba, "Real-time animation of spark discharge," *Visual Computing*, vol. 22, pp. 761–771, Sept. 2006.

[50] R. D. Hill, "Analysis of Irregular Paths of Lightning Channels," *Journal of Geophysical Research*, vol. 73, no. 6, pp. 1897–1906, 1968.

[51] M. J. Berger, *Adaptive mesh refinement for hyperbolic partial differential equations*. PhD thesis, Stanford University, 1982.

[52] M. J. Berger and J. Oliger, "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations," *Journal of Computational Physics*, vol. 53, no. 3, pp. 484–512, 1984.

[53] H. Ji, F.-S. Lien, and E. Yee, "A new adaptive mesh refinement data structure with an application to detonation," *Journal of Computational Physics*, vol. 229, pp. 8981–8993, Nov. 2010.

[54] C. Shen, J.-M. Qiu, and A. Christlieb, "Adaptive mesh refinement based on high order finite difference WENO scheme for multi-scale simulations," *Journal of Computational Physics*, vol. 230, pp. 3780–3802, May 2011.

[55] D. Yoon, H. Kim, and W. Hwang, "Adaptive Mesh Refinement for Weighted Essentially Non-Oscillatory Schemes," *Bulletin of the Korean Mathematical Society*, vol. 45, no. 4, pp. 781–795, 2008.

[56] S. Li and J. M. Hyman, "Adaptive Mesh Refinement for Finite Difference Weno Schemes," tech. rep., Theoretical Division, Los Alamos Laboratory, 2003.

[57] M. J. Berger and R. J. LeVeque, "Adaptive Mesh Refinement Using Wave-Propagation Algorithms for Hyperbolic Systems," *SIAM Journal on Numerical Analysis*, vol. 35, no. 6, pp. 2298–2316, 1998.

[58] M. J. Berger and P. Colella, "Local Adaptive Mesh Refinement for Shock Hydrodynamics," *Journal of Computational Physics*, vol. 82, no. 1, pp. 64–84, 1989.

[59] S. Jain, P. Tsiotras, and H.-M. Zhou, "Adaptive Multiresolution Mesh Refinement for the Solution of Evolution PDEs," *Decision and Control, 2007 46th IEEE Conference*, 2008.

[60] N. Srinivasa, "Adaptive Mesh Refinement for a Finite Difference Scheme Using a Quadtree Decomposition Approach," Master's thesis, Texas A&M University, Nov. 2006.

[61] S. Chandra, X. Li, T. Saif, and M. Parashar, "Enabling scalable parallel implementations of structured adaptive mesh refinement applications," *The Journal of Supercomputing*, vol. 39, pp. 177–203, Mar. 2007.

[62] P. MacNeice and K. Olson, "An overview of the paramesh amr software and some of its applications," in *Adaptive Mesh Refinement-Theory and Applications, Proceedings of the Chicago Workshop on Adaptive Mesh Refinement Methods, Series: Lecture Notes in Computational Science and Engineering* (T. P. T. Linde and G. Weirs, eds.), 41, Berlin: Springer, 2005.

[63] K. Olson, "Paramesh: A parallel adaptive grid tool," in *Parallel Computational Fluid Dynamics 2005: Theory and Applications: Proceedings of the Parallel CFD Conference*, (College Park, MD, U.S.A.), Elsevier, 2006.

[64] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey, "libMesh: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations," *Engineering with Computers*, vol. 22, no. 3–4, pp. 237–254, 2006. http://dx.doi.org/10.1007/s00366-006-0049-3.

[65] S. Morris, "Non-Uniform Load Balancing of Adaptive Mesh Refinement Codes," tech. rep., University of Utah, 2010.

[66] A. M. Wissink, R. D. Hornun, S. R. Kohn, S. S. Smith, and N. Elliott, "Large Scale Parallel Structured AMR Calculations Using the SAMRAI Framework," tech. rep., Lawrence Livermore National Laboratory, 2001.

[67] B. J. Finlayson-Pitts and J. N. Pitts, Jr., *The Atmospheric System*, ch. 2, pp. 15–42. Academic Press, 2000.

[68] A. S. Antoniou, K. I. Karantasis, E. D. Polychronopoulos, and J. A. Ekaterinaris, "Acceleration of a Finite-Difference WENO Scheme for Large-Scale Simulations on Many-Core Architectures," in *Proceedings of the 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, American Institute of Aeronautics and Astronautics, 2010.

[69] H.-Y. Schive, Y.-C. Tsai, and T. Chiueh, "Gamer: a Graphic Processing Unit Accelerated Adaptive-Mesh-Refinement Code for Astrophysics," *The Astrophysical Journal Supplement Series*, vol. 186, pp. 457–484, 2010.

[70] J.-P. Berenger, "A perfectly matched layer for the absorption of electromagnetic waves," *Journal of Computational Physics*, vol. 114, no. 2, pp. 185–200, 1994.

[71] M. Chevalier, *Advances in the perfectly matched layer absorbing boundary condition and a technique for efficiently modeling long path propagation with applications to finite-difference grid techniques.* PhD thesis, Stanford University, 2006.

[72] S. G. Johnson, "Notes on Perfectly Matched Layers (PMLs)," 2010.