A Parallel Implementation of the Discontinuous-Galerkin-Hancock Method on Unstructured Meshes with Adaptive Mesh Refinement

by Vincent Gascon

A thesis submitted to the University of Ottawa in conformity with the requirements for the degree of Master of Applied Science

Department of Mechanical Engineering
University of Ottawa

Abstract

The steady increase in computational power over the last generation gives rise to fields of research that use it, which is why Computational Fluid Dynamics (CFD) has been, and remains, a rapidly evolving field of research. Traditionally, the Navier-Stokes (NS) equations have been used almost exclusively to model viscous gas flows. More recently, the field of moment methods have emerged as an alternative. Moment methods are derived from the kinetic theory of gases. Because of this, they offer certain advantages over the NS model, such as an improved accuracy in describing rarefied gases or multiphase flow, where the microscopic scale can be more prominent. These moment methods are described by first-order hyperbolic partial differential equations (PDE) with stiff local relaxation source terms. The discontinuous-Galerkin (DG) methods, initially designed to solve first order hyperbolic balance laws, are of particular interest in the current study. The discontinuous-Galerkin-Hancock (DGH) method is of third-order accuracy, was designed specifically for the efficient solution of moment methods, and lends itself well to large-scale parallel computing.

Once a scheme like DGH has been identified, it is always coupled with a tessellation of space. In a structured discretization, space is often split into quadrilaterals, which can be numbered in an orderly manner, such as would be considered intuitive or "structured". Contrarily, an unstructured discretization splits space into polygons of theoretically any number of sides. This allows complex geometries to be more easily represented, and hence is more versatile. Given these meshes' unstructured nature, their implementation becomes less straightforward.

The current study outlines a paralellized implementation of the DGH scheme on unstructured meshes in two and three dimensions. An adaptive refinement algorithm was developed alongside this. The implementation allows the computational tasks to be spread amongst many processors, and its adaptive nature allows this implementation to tailor its mesh's detail to the solution that is being computed in real-time.

A multitude of different first-order PDEs in different contexts are solved in the current study. To begin with, a supersonic flow-past-bump case is investigated using the compressible Euler equations. This is done to showcase the adaptive refinement algorithm's power, combined with the scheme's ability to capture strong shocks. Next, the unstructured nature of the implementation is taken advantage of with complex geometries, such as for a multi-element airfoil, and for a room equipped with sound diffusion. The flow around the airfoil is modeled using the 10-moment Gaussian moment closure, to capture viscosity's effects with a first-order hyperbolic model. Finally, the linear convection equations are used to verify the scheme's order of accuracy on 2D and 3D meshes, and the Euler equations are used on a varying number of CPUs to

demonstrate the implementation's scalability on large clusters.

Acknowledgements

I would like to start off by thanking my family and friends for their interest in the success of my career, and constant curiosity of my contribution to the world of research. The success of this thesis would not have been possible without their support. I would like to specifically thank my mother and father for their love and financial support in the completion of both my undergraduate and master's degrees.

The success of this project is also largely due to Professor James McDonald, my research advisor. I thank him for putting some trust in me despite the circumstances of my application. His guidance has proven to be fruitful in the development of this project as well as my career.

Furthermore, I thank the University of Ottawa for providing an environment and opportunities that allowed me to have success in what was the last eight years. Finally, I would like to thank Cassidy Mercier for her support and for pushing me through the tough times that have been the last four years. The global pandemic proved to be a hurdle in more ways than one.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Computational Fluid Dynamics (CFD) is a field of study that has been of increased interest over the past several decades. It allows engineers to minimize performing real-world tests in favour of tests done numerically, which implies many cost and time savings. CFD is not yet a full alternative to real-world tests, however it is becoming increasingly important in the design process. The development of new CFD methods only brings CFD closer to replacing practical tests in many applications. Methods that have minimal computational complexity are required, whilst maintaining high accuracy. The Discontinuous-Galerkin-Hancock (DGH) scheme has both of these properties and is of particular interest in this study. As of yet, this scheme has only been implemented using a structured discretization of space, or structured meshes.

Structured meshes can have an impressive amount of applications despite their structured, and seemingly rigid nature. They can be bent, curved, folded, and much more. This is done to represent an array of different geometries. A great example of this is for airfoils. A structured mesh can be folded around an airfoil into a "C" shape, filling the space without creating low quality meshing anywhere. However, there exist complex enough geometries around which trying to mold a structured mesh decreases the quality of the mesh, or may just be impossible. This is a gap that can be filled with unstructured meshes. Unstructured meshes simply fill any space necessary, often with triangles in 2D or tetrahedra in 3D, and hence will work for any geometry. These meshes are mostly used for geometries that include a lot of detail, or that are considered odd. When looking at the segmented airfoil case, folding a structured mesh around these types of airfoils is more difficult, and can decrease the quality of the mesh in certain crucial areas.

CFD solvers using unstructured meshes are generally slower at computing flows than structured meshes due to their less optimized storage in memory. As the computational complexity of the problem increases, the disparity between the two types of meshes only increases, and is why parallelization is crucial in the unstructured case. The implementation presented in this thesis uses adaptive mesh refinement to allow the unstructured mesh to use the processors at its disposal in parallel. Alonside this, The DGH scheme was designed with parallelization in mind, to allow for a close to linear increase in computing speed relative to number of processors.

1.2 Objective

The first objective of the current study is to implement handling unstructured meshes in a DGH code. Along with this, the second objective is to implement adaptive mesh refinement, which allows the mesh to be refined and partitioned in an adaptive way, making proper use of high computational power when available. This allows big calculations to be made in reasonable amounts of time.

The third objective, naturally, is the verification of the order of accuracy of the DGH scheme on unstructured meshes. This entails doing a convergence study on a simple case for which the exact solution is known.

The final objective is to study the parallel scalability of the implementation. A scalability study is done to observe the implementation's performance as the number of computational cores is increased, hoping for close to a linear relationship between execution time and computational problem size, as well as execution time and number of processors. Currently, this code is confirmed to be third-order accurate with its current implementation for structured meshes.

1.3 Outline of Thesis

The current thesis provides an explanation of moment methods in chapter 2. Following this, a description of the discontinuous-Galerkin-Hancock (DGH) scheme is reviewed in Chapter 3. In Chapter 4, this thesis describes an implementation of the DGH scheme in one, two, and three dimensions. Chapter 4 describes the unstructrured mesh generation software in Section 4.2, the unstructured mesh partitioning software in Section 4.3.5, the unstructured partition implementation itself in Sections 4.3.1 through 4.3.2, and finally, the implementation of refinement for unstructured meshes in 2D and 3D in Section 4.3.5. The final chapter, Chapter 5, shows results obtained using the DGH scheme with unstructured meshes solving many different PDEs, and tests the implementation's order convergence and parallel scalability.

Chapter 2

Moment Methods

2.1 Kinetic Theory

Many fluid dynamics equations and models rely on approximations that see a flow as a general continuum described by one or more continuous fields, in many ways ignoring any dynamics happening at a smaller scale, such as particle-to-particle interactions. Kinetic theory is derived with these particle interactions in mind, using Newton's laws of motion to describe the particle interactions in a mathematical way. However, at scales which are commonly studied, there are too many particles to expect a mathematical description of each and every particle's motion, even given today's computing power. In this case, a statistical description of this group of particles as a whole is considered, keeping particle interactions in mind, to overcome the issue. Kinetic theory of gases, in the context of this study, relies on the following assumptions:

- All matter is composed of discrete particles.
- All particles of a given species are identical.
- Particles are points with no internal structure.
- Particles only exert forces over distances significantly smaller than the mean-free path.
- Only binary collisions occur between particles, meaning a given collision only involves two particles.
- Quantum effects are neglected.
- Colliding particles are statistically uncorrelated.

2.2 The Maxwell-Boltzmann Distribution

A group of particles is statistically described using a probability distribution function, $\mathcal{F}(x_i, v_i, t)$, defined in a phase space composed of physical space and velocity space. This phase-space probability distribution function describes the likelihood that particles at a specific

location have a specific velocity. The Maxwell-Boltzmann distribution is a probability distribution function that describes a gas that is in local thermodynamic equilibrium, which is a state to which a gas will tend given enough time, due to inter-particle collisions. The Maxwell-Boltzmann distribution, or in other words, the Maxwellian probability distribution function, is formulated as

$$\mathcal{M} = n(x_i,t)(\frac{\beta(x_i,t)}{\pi})^{\frac{3}{2}} e^{-\beta(x_i,t)(v_i-u_i)(v_i-u_i)}, \tag{2.1}$$

where

$$\beta(x_i,t) = \frac{m}{2kT(x_i,t)}. \tag{2.2}$$

Here, n is the number of particles, m is the particle mass, u_i is the local average speed of the group of particles, v_i is the speed of a particular particle, k is the Boltzmann constant, $1.3806 \times 10^{-23} \mathrm{J\,K^{-1}}$, and $T(x_i,t)$ is the temperature of the fluid at location, x_i, and time, t. As long as conditions remain static and particle collisions occur, it has been proven that all gases' probability distribution functions will eventually become Maxwellian, given enough time [Gom94].

2.3 Moments of Probability Distribution Functions

Mathematically, moments of probability distribution functions are the function itself multiplied by some weight, w, and integrated over all of velocity space. These moments often correspond to macroscopic quantities, such as density and momentum, which are often the quantities that are desired when finding solutions to fluids problems. A shorthand way of expressing a moment is with angle brackets,

$$\iiint_{-\infty}^{\infty} w(v_i)\mathcal{F}(x_i,v_i,t)\mathrm{d}v_i = \langle w\mathcal{F}\rangle, \tag{2.3}$$

where the angle brackets themselves signify integrating over all of velocity space. The particular macroscopic property obtained from the integration depends on the weight used. For example, the mass density, ρ, can be found by using a weight of $w = m$,

$$\rho = \langle m\mathcal{F}\rangle. \tag{2.4}$$

Similarly, the momentum density can be found by using a weight of $w = mv_i$,

$$\rho u_i = \langle mv_i\mathcal{F}\rangle. \tag{2.5}$$

Here, u_i signifies the bulk velocity of the fluid. It is related to the velocity v_i with $v_i = u_i + c_i$, where c_i is the random velocity. Moments can also be taken using the random velocity, c_i, instead of v_i. For instance, the second-order pressure tensor, P_{ij}, is found using a weight of $w = mc_ic_j$,

$$P_{ij} = \langle mc_ic_j\mathcal{F}\rangle. \tag{2.6}$$

This pressure tensor is the negative of the fluid-stress tensor and is related to the viscous stresses through the expression $P_{ij} = p\delta_{ij} - \sigma_{ij}$, where p is the hydrostatic pressure and σ_{ij} the viscous-stress tensor. The order at which moments can be taken is arbitrary, however most traditional fields will arise from the first few orders, as can be seen with density, momentum.

2.4 The Boltzmann Equation

Fluids that do not find themselves in thermodynamic equilibrium are expected to evolve in time. The Boltzmann equation describes this evolution, but for probability distribution functions. It is given by

$$\frac{\partial \mathcal{F}}{\partial t} + v_i \frac{\partial \mathcal{F}}{\partial x_i} + \frac{\partial a_i \mathcal{F}}{\partial v_i} = \frac{\delta \mathcal{F}}{\delta t}. \tag{2.7}$$

On the left hand side, from left to right, first the term representing the evolution of the probability distribution function in time. The following term represents convection within the fluid, and finally the term describing external forces acting upon the fluid is last. For now, a_i, the term representing accelerations due to external forces, is taken to be zero. The term found on the right hand side, $\frac{\delta \mathcal{F}}{\delta t}$, is the collision operator. It models all the effects the particle collisions have on the evolution of the distribution function, and so it is a core element of kinetic theory. Given that the collision operator's role is to model the collision of all particles within the group, it is a complex term. This is why approximations are mostly used in its place. The BGK operator [BGK54] is one of the most common approximations used, and is the one used in this work. It is written as

$$\frac{\delta \mathcal{F}}{\delta t} = -\frac{\mathcal{F}(x_i, v_i, t)}{\tau_{\mathcal{F}}(x_i, t)} + \frac{\mathcal{M}(x_i, v_i, t)}{\tau_{\mathcal{M}}(x_i, t)}. \tag{2.8}$$

This essentially removes non-equilibrium $\mathcal{F}$ particles at a rate dictated by $\tau_{\mathcal{F}}$, and adding equilibrium $\mathcal{M}$ particles at a speed dictated by $\tau_{\mathcal{M}}$. In order to conserve mass, particles being added are paired one-by-one with those being removed, and so particles should be added at the same speed as they are removed. This means $\tau_{\mathcal{F}} = \tau_{\mathcal{M}} = \tau$.

$$\frac{\delta \mathcal{F}}{\delta t} = -\frac{\mathcal{F}(x_i, v_i, t) - \mathcal{M}(x_i, v_i, t)}{\tau} \tag{2.9}$$

2.4.1 Maxwell's Equation of Change

The Boltzmann equation is very high dimensional and provides a huge amount of information about gas-particle motions that is often unimportant. Equations governing the evolution of moments are often more useful. Assuming zero external forces, and then taking moments of the Boltzmann equation, Maxwell's equation of change is obtained,

$$\frac{\partial}{\partial t}\langle mw\mathcal{F}\rangle + \frac{\partial}{\partial x_i}\langle mv_i w\mathcal{F}\rangle = \langle mw\frac{\delta \mathcal{F}}{\delta t}\rangle, \tag{2.10}$$

which can prove to be more useful. Though Maxwell's equation of change provides a balance law for the evolution of a single moment, often multiple fields are of interest. One can define a set of weights, $\boldsymbol{w} = [w_0, w_1, ..., w_N]^T$, that results in a vector of moments, $U = \langle m\boldsymbol{w}\mathcal{F}\rangle$.

The flux dyad, $F_i = \langle m v_i \boldsymbol{w} \mathcal{F} \rangle$, and the local source term, $S = \langle m \boldsymbol{w} \frac{\delta \mathcal{F}}{\delta t} \rangle$, can also be defined. Substituting this into Equation (2.10) yields

$$\frac{\partial U}{\partial t} + \frac{\partial F_i}{\partial x_i} = S. \tag{2.11}$$

Here, choosing U, F, and S leads to a collection of first-order PDEs that describe the evolution of multiple macroscopic properties of a fluid. Examples of these are covered in Section 2.5.

2.5 Moment Closure

Equation (2.11), obtained in the last section, is actually a general form for a set of first-order PDEs that describe fluid macroscopic properties. However, the flux dyad and the source term can not generally be uniquely specified as a function of variables in the solution vector $\boldsymbol{U}$. The flux dyad ends up always having moments that are one order higher than those found in the solution vector, and so the system remains "open". To overcome this problem, a form for $\mathcal{F}$ can be proposed, which is a function of free parameters, $\boldsymbol{\alpha} = [\alpha_1, ...\alpha_N]$, the number of which is identical to the number of entries in the $\boldsymbol{U}$ vector. This allows $\mathcal{F}$ to depend on the solution vector $\boldsymbol{U}$, and "close" the system. This is the general principle behind the concept of moment closure, originally proposed by Grad [Gra49]. The critical step of choosing the specific form of $\mathcal{F}$ remains.

Originally, Grad chose $\mathcal{F}$ such that

$$\mathcal{F} = \mathcal{M}(\alpha^T W). \tag{2.12}$$

While this does close the system, it cannot produce a solution for many CFD problems. This is simply a product between a polynomial and the Maxwellian distribution described in Section 2.2, and does not necessarily result in a strictly positive distribution. More devastating still, the flux Jacobian can develop complex eigenvalues. The hyperbolicity of the first-order balance laws describing the model is lost and the system becomes ill-posed for initial-value problems. Since Grad's work, other moment closures have been developed, their individual properties stemming from specific choices of forms for $\mathcal{F}$.

2.5.1 Maximum-Entropy Closures

Maximizing entropy is a concept that was explored to find a better form for $\mathcal{F}$. Maximizing entropy is tied heavily with real physical situations, given that all particle interactions tend to increase entropy. Finding the form for $\mathcal{F}$ that maximizes entropy while remaining consistent with moment constraints follows this philosophy, which is a great argument for this model. Boltzmann proved that the entropy density for a non-equilibrium gas can be found as

$$S = \langle -k\mathcal{F} \ln \frac{\mathcal{F}}{y} \rangle, \tag{2.13}$$

where y is an arbitrary constant. This entropy, S, is also the result of taking the moment $\langle mw\mathcal{F}\rangle$ with velocity weight

$$w = -k\left(\ln\frac{\mathcal{F}}{z} - 1\right), \tag{2.14}$$

where $y = ez$. After some algebra, maximizing the entropy, S, boils down to maximizing the expression

$$\max\langle -k\mathcal{F}\ln\frac{\mathcal{F}}{z} + k\mathcal{F}\rangle, \tag{2.15}$$

while satisfying moments $U = \langle mw\mathcal{F}\rangle$ with velocity weights w. This maximization problem can be solved using Lagrange multipliers, $\boldsymbol{\alpha}$. This converts the problem to a search for critical points of a function, I, written as

$$I = \langle -k\mathcal{F}\ln\frac{\mathcal{F}}{z} + k\mathcal{F}\rangle - \boldsymbol{\alpha}^{\boldsymbol{T}}(U - \langle W\mathcal{F}\rangle). \tag{2.16}$$

At a point where the function I's value is at an extremum, $\frac{dI}{d\mathcal{F}} = 0$, one finds

$$\frac{dI}{d\mathcal{F}} = \langle -k\ln\frac{\mathcal{F}}{z}\rangle - \alpha\langle W\rangle \tag{2.17}$$

$$0 = \langle \ln\mathcal{F} - \alpha W\rangle. \tag{2.18}$$

As a reminder, the beginning goal was to find the form for $\mathcal{F}$ that would maximize entropy. In this case, it is clear that the form for $\mathcal{F}$ that satisfies Equation (2.18) is

$$\mathcal{F} = e^{\alpha^T W}. \tag{2.19}$$

Given that now $\alpha^T W$ is in an exponential, contrary to Grad's original closure, $\mathcal{F}$ is guaranteed to be positive. This means it will always predict a positive number of particles at any location, which is, once again, consistent with real physical situations. Levermore provides a list of requirements on the velocity weights present in W that lead to well-behaved closures [Lev96a]. The two largest-order examples of this type of closure are the Euler equations and the ten-moment equations. The Euler equations use weights $W_5 = [m, mv_i, mv_iv_i]^T$, and the ten-moment equations use weights $W_{10} = [m, mv_i, mv_iv_j]^T$. They are the most relevant to this thesis, and are described in more detail later in this work.

Chapter 3

Discontinuous-Galerkin Hancock Method

3.1 Introduction

The crux of CFD's efficacy boils down to the method it uses to numerically solve relevant PDEs. A common practice when developing CFD methods is to discretize fields that are, in reality, continuous in space and time. Space is often discretized using the finite-volume method, but it is interesting to take an alternate approach. The discontinuous-Galerkin (DG) method is a finite-element method, created by Reed and Hill [RH73], initially meant to be used for solving the steady linear neutron transport equation. Many error analyses were done on this method, first LeSaint and Raviart showing that its rate of convergence, for a DG method of degree k, is $(\Delta x)^{k+1}$ for cartesian grids [LR74]. Next, Johnson and Pitkaranta proved that the rate of convergence for a DG method of degree k is $(\Delta x)^{k+\frac{1}{2}}$ for general triangulations [JP86].

After the DG method had proven to be worth investing time in its research, a few methods were developed, which use the DG discretization, but initially in space only. These semi-discrete methods used the method-of-lines (MOL), developed by Schiesser [Sch91], to separate the spatial and temporal discretization. The MOL approximates a PDE with a system of ODEs, which forces the application of severe limits on time-stepping given stability issues. An example of this type of model is the Runge-Kutta discontinuous-Galerkin (RKDG) method [CS98]. Following this, fully discrete methods for space-time discretization were developed, having in mind the goal of eliminating stability issues. A few were developed, but proved to have similar stability issues.

The discontinuous-Galerkin-Hancock (DGH) scheme is a finite-element method developed by Suzuki and Van Leer [Suz08]. Is is a coupled space-time method. It is used to solve hyperbolic balance laws, and it was initially conceived to be an efficient method for the solution of PDEs resulting from moment methods, as described in Chapter 2. The DG method was interesting to Suzuki and Van Leer given its compactness in space and time. Compactness refers to a cell's lack of dependance on a large group of neighboring cells in order to compute an update. In this case, a cell's update depends only on itself and immediate neighbours. Compactness is an

attractive feature for parralel computing, given the minimal communication needed between cells. The DGH scheme is well suited for parallel implementation on large computers, given the low requirement for communication. It is also tailored to work well with PDEs that have stiff source terms. This scheme is third-order accurate in both space and time on structured meshes.

3.2 Weak Formulation

Partial differential equations for which the DGH scheme provides solutions have the form

$$\frac{\partial U}{\partial t} + \frac{\partial F_i}{\partial x_i} = S\,. \tag{3.1}$$

In this method, the solution U is approximated as the weighted sum of simple functions. This trial solution is of the form

$$U = \sum_{j=0}^{n} \alpha_j \phi_j, \;\; U \in \mathbb{R}^n\,, \tag{3.2}$$

where the ϕ_j are the basis functions and the α_j are weights or the degrees of freedom of the scheme. The DGH method makes use of a linear representation of the solution within each element, and gets its "discontinuous" nature from the fact that each basis function only exists within a single cell. They are zero everywhere else. The trial solution is therefore discontinuous across cell boundaries. The basis functions and weights in each cell are defines as

$$\begin{aligned} \alpha_0 &= \overline{U} & \qquad \phi_0 &= 1 \\ \alpha_1 &= \frac{\partial U}{\partial x} & \qquad \phi_1 &= (x - x_c) \\ \alpha_2 &= \frac{\partial U}{\partial y} & \qquad \phi_2 &= (y - y_c) \\ \alpha_3 &= \frac{\partial U}{\partial z} & \qquad \phi_3 &= (z - z_c) \end{aligned}\,,$$

where $\overline{U}$ denotes the average solution within the cell. It is worth noting here that the basis functions within the cell do not depend on time, which will be useful later. This "trial solution" translates to a 1st-order polynomial representation of the solution within a cell, which is illustrated in Figure 3.1. This increases the scheme's spatial accuracy. Typically, schemes with a linear solution representation attain 2nd-order accuracy in space. Here, the constants $[x_c, y_c, z_c]$ represent the centroid of the cell in question. Since this scheme is of Galerkin type, the basis functions are chosen to be the test functions. A solution is sought such that taking the product of PDEs of the type shown in Equation (3.1) with the test functions, ϕ_i, and integrating over the space-time domain, $\Omega(t) \times T$, results in a satisfied equality. Given that the test functions are non-zero only within a cell, k, and do not change over time interval, n, the space integral can be specified to be bounded within the space of a single element, Ω_k. The time range for T will also be specified to be a single time step, $[t^n, t^{n+1}]$. The resulting equation can be written

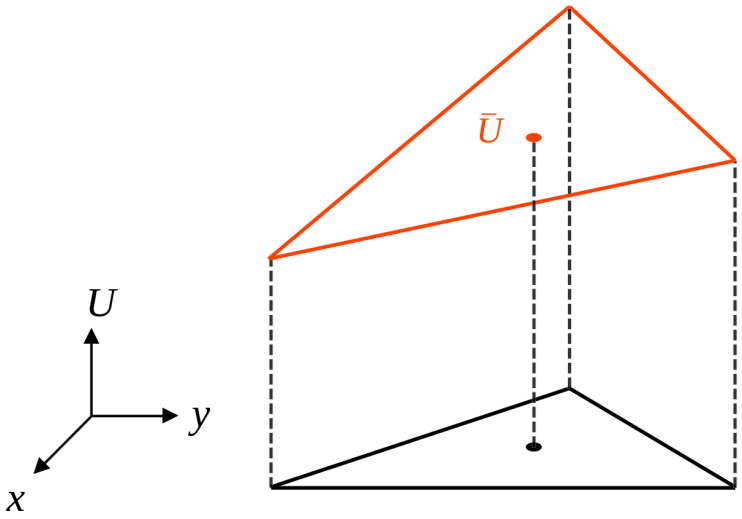

Figure 3.1: A visualization of the polynomial representation of the solution within a two-dimensional triangular cell, with the solution U plotted on the vertical dimension.

as

$$\underbrace{\int_{t^n}^{t^{n+1}}\int_{\Omega_k} \phi_j \frac{\partial U}{\partial t}\, dx_i\, \mathrm{d}t}_{\text{Solution Term}} + \underbrace{\int_{t^n}^{t^{n+1}}\int_{\Omega_k} \phi_i \frac{\partial F_i}{\partial x_i}\, \mathrm{d}x_i\, \mathrm{d}t}_{\text{Flux Term}} = \underbrace{\int_{t^n}^{t^{n+1}}\int_{\Omega_k} \phi_j S\, \mathrm{d}x_i\, \mathrm{d}t}_{\text{Source Term}} . \tag{3.3}$$

3.2.1 Integral Evaluations

Solution Term

To start off, performing integration by parts on the time integral of the solution term is done. Given the test functions' lack of dependence on time, terms can be eliminated, resulting in

$$\int_{t^n}^{t^{n+1}}\int_{\Omega_k} \phi_j \frac{\partial U}{\partial t}\, \mathrm{d}x_i\, \mathrm{d}t = \left[\int_{\Omega_k} U\phi_j\, \mathrm{d}x_i\right]_{t_n}^{t_{n+1}} - \cancel{\int_{t^n}^{t^{n+1}}\int_{\Omega_k} U\frac{\partial \phi_j}{\partial t}\mathrm{d}x_i\, \mathrm{d}t} ,$$

$$\int_{t^n}^{t^{n+1}}\int_{\Omega_k} \phi_j \frac{\partial U}{\partial t}\, \mathrm{d}x_i\, \mathrm{d}t = \int_{\Omega_k} U^{n+1}\phi_j\, \mathrm{d}x_i - \int_{\Omega_k} U^n\phi_j\, \mathrm{d}x_i . \tag{3.4}$$

Flux Term

Next, using the divergence theorem, the flux-term's space integral can be split, resulting in

$$\int_{t^n}^{t^{n+1}}\int_{\Omega_k} \phi_j \frac{\partial F_i}{\partial x_i}\, \mathrm{d}x_i\, \mathrm{d}t = \int_{t^n}^{t^{n+1}}\int_{\Gamma_k} \phi_j F_i \cdot \hat{n}_i\, d\Gamma_k\, \mathrm{d}t - \int_{t^n}^{t^{n+1}}\int_{\Omega_k} F_i \frac{\partial \phi_j}{\partial x_i}\, \mathrm{d}x_i\, \mathrm{d}t , \tag{3.5}$$

where Γ_k is the boundary of cell k, and $\hat{n}_i$ is the outward-pointing unit normal to the boundary.

3.2.2 Final Weak Form

Inserting Equations (3.5), (3.4), as well as Equation (3.2), into Equation (3.3) leads to

$$\int_{\Omega_k} \phi_j \left[U^{n+1} - U^n\right] \, \mathrm{d}x_i = \int_{t^n}^{t^{n+1}} \int_{\Omega_k} F_i \frac{\partial \phi_j}{\partial x_i} \, \mathrm{d}x_i \, \mathrm{d}t - \int_{t^n}^{t^{n+1}} \int_{\Gamma_k} \phi_j F_i \cdot \hat{n}_i \, d\Gamma_k \, \mathrm{d}t + \int_{t^n}^{t^{n+1}} \int_{\Omega_k} \phi_j S \, \mathrm{d}x_i \, \mathrm{d}t \,. \quad (3.6)$$

3.3 Update Formulas – α_0

To obtain the update formula for a particular degree of freedom, α_i, the projection of the trial solution on to the associated basis function must be completed. For example, to update the average solution, $\alpha_0 = \overline{U}$, the trial solution must be projected onto the basis function $\phi_0 = 1$ using Equation (3.6). In this case, the result is

$$[\alpha_0]_k^{n+1} = [\alpha_0]_k^n - \frac{1}{V_k} \underbrace{\int_{t^n}^{t^{n+1}}}_{A} \underbrace{\int_{\Gamma_k}}_{B} F_i \cdot \hat{n}_i \, d\Gamma_k \, \mathrm{d}t + \frac{1}{V_k} \underbrace{\int_{t^n}^{t^{n+1}}}_{C} \underbrace{\int_{\Omega_k}}_{D} S \, \mathrm{d}x_i \, \mathrm{d}t \,, \quad (3.7)$$

where letters "A","B","C", and "D" designate the method by which these integrals will be approximated.

$A \rightarrow$ midpoint

$B \rightarrow$ Gaussian quadrature (2D:2pt, 3D:6pt)

$C \rightarrow$ Radau IIA

$D \rightarrow$ Gaussian quadrature (1pt)

Also, V_k is the volume of the cell, or in two dimensions, the area of the cell.

3.3.1 Time Integration

Radau IIA

Suzuki [Suz08] chose to use the Radau IIA method to approximate the source term time integral (integral "C" in Equation (3.7)). The Radau IIA method is a method for obtaining the numerical solution of ordinary differential equations of the form

$$\frac{dy}{dt} = f(t, y) \,. \quad (3.8)$$

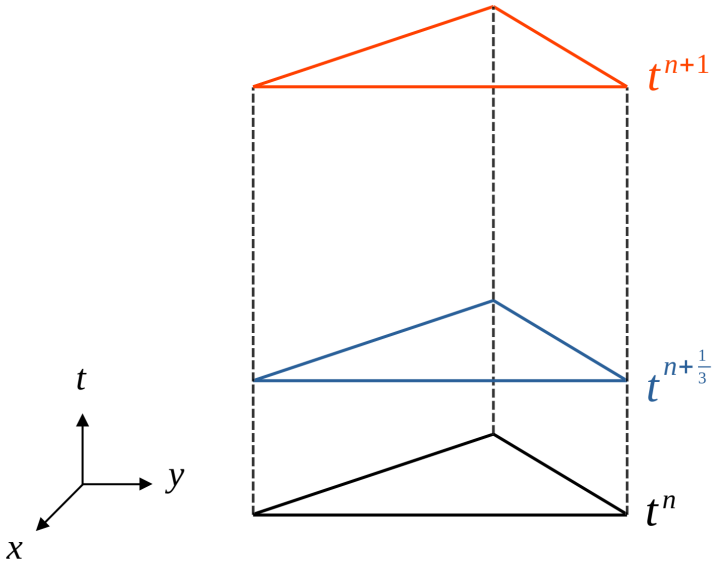

Figure 3.2: A visualization of the fractional time step $t^{n+\frac{1}{3}}$ (blue), by plotting the time dimension perpendicular to a two-dimensional triangular element.

The Radau IIA method involves incorporating a fractional time step at time $t^{n+\frac{1}{3}}$. This means numerical time steps for Equation (3.8) are

$$y^{n+\frac{1}{3}} = y^n + \Delta t \left[\frac{5}{12} \frac{dy}{dt}^{n+\frac{1}{3}} - \frac{1}{12} \frac{dy}{dt}^{n+1} \right], \tag{3.9}$$

$$y^{n+1} = y^n + \Delta t \left[\frac{3}{4} \frac{dy}{dt}^{n+\frac{1}{3}} + \frac{1}{4} \frac{dy}{dt}^{n+1} \right]. \tag{3.10}$$

Figure 3.2, a reference 2D element projected into the time dimension, shows a visual representation of the fractional time step.

Hancock's Predictor Step

The flux-term time integral (Integral "A" from Equation (3.7)) is approximated using midpoint integration. This simply implies that the integral is evaluated using the flux-term at the halfway point of the targeted point in time. This extra step allows the scheme to be of higher-order accuracy in time. Since with Radau IIA, a coupled set of equations that requires solving solutions at $t^{n+\frac{1}{3}}$ and t^{n+1} is created, midpoint integration has the intercell fluxes being evaluated at quadrature points that are at $t^{n+\frac{1}{6}}$ and $t^{n+\frac{1}{2}}$. These fluxes are given by the approximate solution of a Riemann problem between the solution in the current cell and the neighbouring cell. To find the solution in the current cell k at a particular quadrature point h, and fractional time $t^{n+\epsilon}$, the average solution $\overline{U}$ is advanced by using Equation (3.7),

$$\overline{U}_k^{n+\epsilon} = \overline{U}_k^n - \frac{1}{V_k} \iint\limits_{\Gamma_k \times t} F_i \cdot \hat{n}_i \, d\Gamma_k \, \mathrm{d}t + \frac{1}{V_k} \iint\limits_{\Omega_k \times t} S \, \mathrm{d}x_i \, \mathrm{d}t \, . \tag{3.11}$$

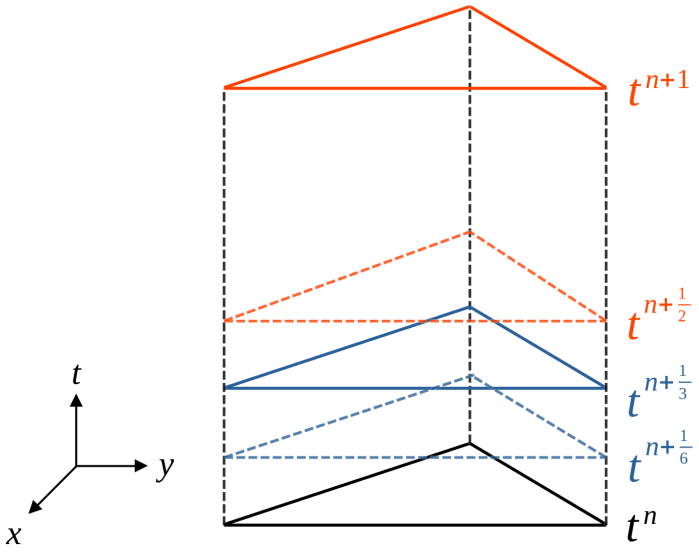

Figure 3.3: A visualization of the additional fractional time steps imparted by the midpoint integral approximation, by plotting the time dimension perpendicular to a triangular element.

Following this, the solution at the edge of the cell can be found using the the known current slope values. This is Hancock's predictor step

$$U_h = \gamma_k^n \underbrace{\begin{bmatrix} \frac{\partial U_k}{\partial x} \\ \frac{\partial U_k}{\partial y} \\ \frac{\partial U_k}{\partial z} \end{bmatrix} \begin{bmatrix} x - x_{ck} \\ y - y_{ck} \\ z - z_{ck} \end{bmatrix}}_{Trial\ Solution} + \underbrace{\overline{U}_k^n - \frac{1}{V_k} \iint_{\Gamma_k \times t} F_i \cdot \hat{n}_i \, d\Gamma_k \, \mathrm{d}t + \frac{1}{V_k} \iint_{\Omega_k \times t} S \, \mathrm{d}x_i \, \mathrm{d}t}_{\overline{U}_k^{n+\epsilon}} , \tag{3.12}$$

where γ_k^n is a vector of slope limiter values. These limits are necessary to guarantee monotonicity, and to prevent oscillations in high-gradient regions. The slope limiter used for this thesis is the Venkatakrishnan limiter [Ven95]. Equation (3.12) is fully local, so all the information needed is already within the cell; no communication between cells occurs during the Hancock predictor step. Using the same space-time representation as in Figure 3.2, Figure 3.3 illustrates the two extra fractional time steps.

3.3.2 Space Integration

Spacial integrations from Equation (3.7) (integrals "B" and "D") are completed using Gaussian quadrature. The volume integral from the flux term of Equation (3.18), also uses Gaussian quadrature. Gaussian quadrature states that the integral of a function over the domain is approximately equal to the sum of the weights times the value of the function at specified quadrature points,

$$\int_a^b f(x) \, \mathrm{d}x \approx \sum_{i=1}^{n} w_i f(x_i) \, .$$

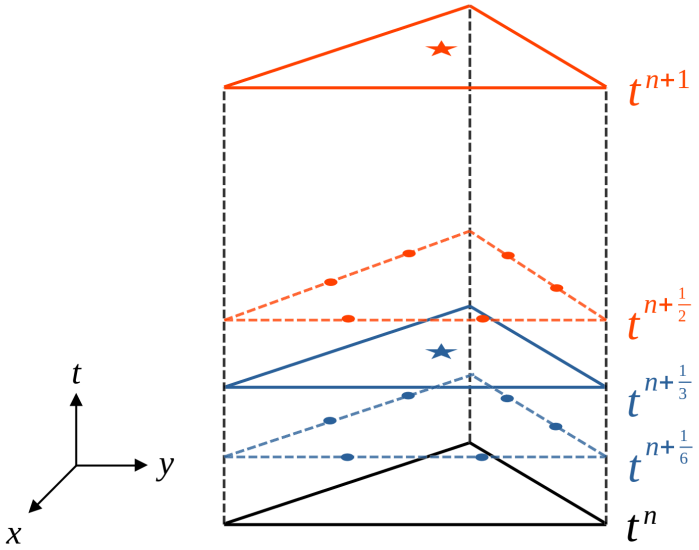

Figure 3.4: A visualization of the volume quadrature points on the 2D space-time element. The circular quadrature points are used for dual integral "A" and "B" in Equation (3.7). The star quadrature points are used for evaluating the Source term Integral in both Equation (3.12) and dual integral "C" and "D" in Equation (3.7).

This equality becomes exact so long as the function being evaluated is a polynomial of degree $(2n-1)$ or less, where n is the number of quadrature points chosen per dimension. The locations of the quadrature points are illustrated on the same space-time reference element in Figures 3.2 and 3.3.

3.3.3 Final Update Formula – α_0

The source term from Equation (3.7), after approximating integrals "C" and "D" for states $n+1$ and $n+\frac{1}{3}$, results in

$$\underline{n+1:} \quad \int_{t^n}^{t^{n+1}} \int_{\Omega_k} S \, \mathrm{d}x_i \, \mathrm{d}t = \Delta t \sum_Q w_Q \left[\frac{3}{4} S^{n+\frac{1}{3}} + \frac{1}{4} S^{n+1} \right], \tag{3.13}$$

$$\underline{n+\frac{1}{3}:} \quad \int_{t^n}^{t^{n+1}} \int_{\Omega_k} S \, \mathrm{d}x_i \, \mathrm{d}t = \Delta t \sum_Q w_Q \left[\frac{5}{12} S^{n+\frac{1}{3}} - \frac{1}{12} S^{n+1} \right]. \tag{3.14}$$

Here, "w" is the quadrature weight and "Q" is the index that goes through all quadrature points (in the previous space-time representation's 2D case). For the source term's case, there is only one quadrature point in the middle of the cell. Now, the flux term double integral (integrals "A" and "B" from Equation (3.7)), after approximating its integrals using Gaussian quadrature

for space and midpoint for time, at states $n+1$ and $n+\frac{1}{3}$, results in

$$\underline{n+1:} \qquad \int_{t^n}^{t^{n+1}} \int_{\Gamma_k} F_i \cdot \hat{n}_i \, d\Gamma_k \, \mathrm{d}t = \Delta t \sum_Q w_Q F^{n+\frac{1}{2}} \tag{3.15}$$

$$\underline{n+\frac{1}{3}:} \qquad \int_{t^n}^{t^{n+\frac{1}{3}}} \int_{\Gamma_k} F_i \cdot \hat{n}_i \, d\Gamma_k \, \mathrm{d}t = \frac{\Delta t}{3} \sum_Q w_Q F^{n+\frac{1}{6}} \, . \tag{3.16}$$

In this case, "Q" is the index that goes through all flux quadrature points on the boundary of the cell. Here, $F_i \cdot \hat{n}_i$ is found by using an approximate Riemann solver between cell k and its neighbour using U from Equation (3.12) as its input. However, $\hat{n}_i$ has still not been dealt with. To do this, the Riemann problem is solved in a rotated frame of reference, where the cell's interface is vertical, and $\hat{n}_i$ is horizontal. Fluxes are hence obtained in a rotated frame of reference, which must then be rotated back to the original frame of reference to be accounted for in each neighbouring cell. Equations (3.13), (3.14), (3.15), and (3.16) can be substituted into the original update formula Equation (3.7) to obtain

$$\begin{bmatrix} \alpha_0^{n+\frac{1}{3}} \\ \alpha_0^{n+1} \end{bmatrix} = \begin{bmatrix} \alpha_0^{n} \\ \alpha_0^{n} \end{bmatrix} - \frac{\Delta t}{V_k} \begin{bmatrix} \frac{1}{3} \sum_Q w_Q F_Q^{n+\frac{1}{6}} \\ \sum_Q w_Q F_Q^{n+\frac{1}{2}} \end{bmatrix} + \Delta t \begin{bmatrix} \frac{5}{12} & -\frac{1}{12} \\ \frac{3}{4} & \frac{1}{4} \end{bmatrix} \begin{bmatrix} S^{n+\frac{1}{3}} \\ S^{n+1} \end{bmatrix} \, . \tag{3.17}$$

3.4 Update Formulas - Slopes α_1 - α_3

As for the other degrees of freedom $[\alpha_1 = \frac{\partial U}{\partial x}, \alpha_2 = \frac{\partial U}{\partial y}, \alpha_3 = \frac{\partial U}{\partial z}]$, just like for α_0, the projection of the trial solution must be completed on the associated basis function. In this case however, $[\phi_1 = (x - x_c), \phi_2 = (y - y_c), \phi_3 = (z - z_c)]$ must be used alongside Equation (3.3) to complete the projection. This results in

$$\begin{aligned} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}_k^{n+1} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}_k^{n} + K_k \Bigg(- \underbrace{\int_{t^n}^{t^{n+1}}}_{A} \underbrace{\int_{\Gamma_k}}_{B} \begin{bmatrix} x - x_c \\ y - y_c \\ z - z_c \end{bmatrix}_k F_i \cdot \hat{n}_i \, d\Gamma_k \, \mathrm{d}t \; + \underbrace{\int_{t^n}^{t^{n+1}}}_{C} \underbrace{\int_{\Omega_k}}_{D} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_k dx_i \, dt \\ + \underbrace{\int_{t^n}^{t^{n+1}}}_{E} \underbrace{\int_{\Omega_k}}_{F} \begin{bmatrix} x - x_c \\ y - y_c \\ z - z_c \end{bmatrix}_k S \, dx_i \, dt \Bigg) \, . \end{aligned} \tag{3.18}$$

where letters "A","B","C","D","E", and "F" designate the method by which these integrals are approximated.

$$
\begin{aligned}
A &\rightarrow \text{midpoint} \\
B &\rightarrow \text{Gaussian quadrature (2D:2-point, 3D:6-point)} \\
C &\rightarrow \begin{cases} \text{Radau IIA}, & \text{for } \alpha^{n+1} \\ \text{Trapezoid}, & \text{for } \alpha^{n+\frac{1}{3}} \end{cases} \\
D &\rightarrow \text{Gaussian quadrature (2D:6-point, 3D:10-point)} \\
E &\rightarrow \text{Radau IIA} \\
F &\rightarrow \text{Gaussian quadrature (1pt)}
\end{aligned}
$$

Finally K_k represents the inverse of the element moment-of-inertia matrix. In three-dimensions, it is given by

$$
K_k = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}^{-1} , \tag{3.19}
$$

where the contents of the matrix represent the moments of inertia of cell k.

3.4.1 Time Integration

The slopes' update formula uses all the same types of time integration as the average solution, such as midpoint and Radau IIA. However, in this case, to obtain Radau IIA's fractional time step at $t^{n+\frac{1}{3}}$, the trapezoidal integration method is used, as opposed to midpoint. Update formula (3.18) contains a trapezoidal time integral (Integral "C" in Equation (3.18)). Since the trapezoid method is used to integrate only up to $\alpha^{n+\frac{1}{3}}$, instead of using the value at $\alpha^{n+\frac{1}{6}}$, half of α^n is added with half of $\alpha^{n+\frac{1}{3}}$.

3.4.2 Space Integration

Equation (3.7) contains a volume integral of only the source term (integral "D" of Equation (3.7)). In this scheme, everywhere there is a volume integral of the source term, it is integrated using 1-point quadrature at the centroid. However, Equation (3.18) also contains a volume integral of a flux term. In this case a slightly better quadrature rule is needed. A 6-point quadrature rule is used in two dimensions, and a 10-point quadrature rule is be used in three dimensions. Figure 3.5 illustrates the points for the two-dimensional case.

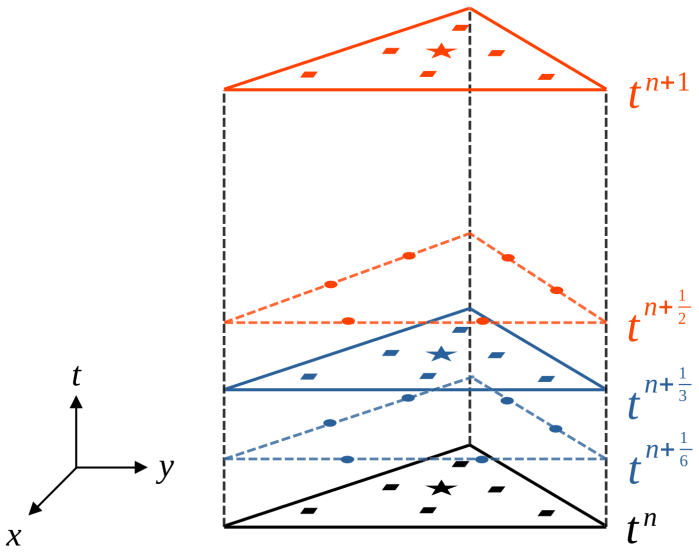

Figure 3.5: A visualization of the volume quadrature points on the 2D space-time element pictured in 3.4. The circular quadrature points, in this case, are used for dual integral "A" and "B" in Equation (3.18). The star quadrature points are used for evaluating the Source term dual Integral 'E" and "F" in Equation (3.18). The square quadrature points are used for the dual integral "C" and "D" in Equation (3.18).

3.4.3 Final Update Formula - α_1-α_3

The first flux term from Equation (3.18), the one with integrals "A" and "B", after approximating the integrals, becomes:

$$\underline{n+1:} \quad \int_{t^n}^{t^{n+1}} \int_{\Gamma_k} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix}_k F_i \cdot \hat{n}_i \, d\Gamma_k \, \mathrm{d}t = \Delta t \sum_Q \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix}_k w_Q F^{n+\frac{1}{2}} \tag{3.20}$$

$$\underline{n+\frac{1}{3}:} \quad \int_{t^n}^{t^{n+\frac{1}{3}}} \int_{\Gamma_k} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix}_k F_i \cdot \hat{n}_i \, d\Gamma_k \, \mathrm{d}t = \Delta t \sum_Q \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix}_k w_Q F^{n+\frac{1}{6}} \tag{3.21}$$

Here, w is the quadrature weight and Q is the index that counts through all quadrature points. In the two-dimensional case, 2 points on each boundary are needed, for a total of 6 points per fractional time step. In three dimensions, there are 6 points on each face, and there are 4 faces, for a total of 24 points per fractional time step. Here, F is found with an approximate Riemann solver between cell k and its neighbour using U from Equation (3.12) as its input. This value was found for the α_0 update, and can be reused here.

The second flux term from Equation (3.18) containing integrals "C" and "D", after approximating the integrals, becomes:

$$\underline{n+1:} \qquad \int_{t^n}^{t^{n+1}} \int_{\Omega_k} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_k \mathrm{d}x_i \, \mathrm{d}t = \Delta t \sum_Q w_Q \left[\frac{3}{4} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_k^{n+\frac{1}{3}} + \frac{1}{4} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_k^{n+1} \right] \tag{3.22}$$

$$\underline{n+\frac{1}{3}:} \qquad \int_{t^n}^{t^{n+\frac{1}{3}}} \int_{\Omega_k} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_k \mathrm{d}x_i \, \mathrm{d}t = \Delta t \sum_Q w_Q \left[\frac{1}{2} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_k^{n} + \frac{1}{2} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_k^{n+\frac{1}{3}} \right] \tag{3.23}$$

where w is the quadrature weight and Q is the index that goes through all quadrature points. In the two-dimensional case, 6 points on each fractional time step exist. In three dimensions, there are 10 points on each fractional time step.

Finally, the source term from Equation (3.18) containing integrals "E" and "F", after approximating the integrals, becomes

$$\underline{n+1:} \qquad \int_{t^n}^{t^{n+1}} \int_{\Omega_k} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix}_k S \, \mathrm{d}x_i \, \mathrm{d}t = I_{ijk} \Delta t \left[\frac{3}{4} \frac{\partial S}{\partial U}^{n+\frac{1}{3}}_{x_c} + \frac{1}{4} \frac{\partial S}{\partial U}^{n+1}_{x_c} \right], \tag{3.24}$$

$$\underline{n+1:} \qquad \int_{t^n}^{t^{n+1}} \int_{\Omega_k} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix}_k S \, \mathrm{d}x_i \, \mathrm{d}t = I_{ijk} \Delta t \left[\frac{5}{12} \frac{\partial S}{\partial U}^{n+\frac{1}{3}}_{x_c} - \frac{1}{12} \frac{\partial S}{\partial U}^{n+1}_{x_c} \right]. \tag{3.25}$$

Now that Equations (3.20), (3.21), (3.22), (3.23), (3.24), and (3.25) are formulated, they can be substituted into the original update formula (3.18) to obtain

$$
\begin{bmatrix} \alpha_1^{n+\frac{1}{3}} \\ \alpha_2^{n+\frac{1}{3}} \\ \alpha_3^{n+\frac{1}{3}} \\ \alpha_1^{n+1} \\ \alpha_2^{n+1} \\ \alpha_3^{n+1} \end{bmatrix} = \begin{bmatrix} \alpha_1^{n} \\ \alpha_2^{n} \\ \alpha_3^{n} \\ \alpha_1^{n} \\ \alpha_2^{n} \\ \alpha_3^{n} \end{bmatrix} - \Delta t \begin{bmatrix} K_i & 0 & 0 \\ 0 & K_j & 0 \\ 0 & 0 & K_k \end{bmatrix} \begin{bmatrix} \frac{1}{3}\sum w\phi_1 F^{n+\frac{1}{6}} \\ \frac{1}{3}\sum w\phi_2 F^{n+\frac{1}{6}} \\ \frac{1}{3}\sum w\phi_3 F^{n+\frac{1}{6}} \\ \sum w\phi_1 F^{n+\frac{1}{2}} \\ \sum w\phi_2 F^{n+\frac{1}{2}} \\ \sum w\phi_3 F^{n+\frac{1}{2}} \end{bmatrix}
$$

$$
+ \Delta t \begin{bmatrix} K_i & 0 & 0 \\ 0 & K_j & 0 \\ 0 & 0 & K_k \end{bmatrix} \begin{bmatrix} \sum w(\frac{1}{2}f^n + \frac{1}{2}f^{n+\frac{1}{3}}) \cdot \hat{i} \\ \sum w(\frac{1}{2}f^n + \frac{1}{2}f^{n+\frac{1}{3}}) \cdot \hat{j} \\ \sum w(\frac{1}{2}f^n + \frac{1}{2}f^{n+\frac{1}{3}}) \cdot \hat{k} \\ \sum w(\frac{3}{4}f^n + \frac{1}{4}f^{n+1}) \cdot \hat{i} \\ \sum w(\frac{3}{4}f^n + \frac{1}{4}f^{n+1}) \cdot \hat{j} \\ \sum w(\frac{3}{4}f^n + \frac{1}{4}f^{n+1}) \cdot \hat{k} \end{bmatrix}
$$

$$
+ \Delta t \begin{bmatrix} \frac{5}{12} & 0 & 0 & \frac{-1}{12} & 0 & 0 \\ 0 & \frac{5}{12} & 0 & 0 & \frac{-1}{12} & 0 \\ 0 & 0 & \frac{5}{12} & 0 & 0 & \frac{-1}{12} \\ \frac{3}{4} & 0 & 0 & \frac{1}{4} & 0 & 0 \\ 0 & \frac{3}{4} & 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{3}{4} & 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} \frac{\partial S}{\partial U}^{n+\frac{1}{3}} \alpha_1^{n+\frac{1}{3}} \\ \frac{\partial S}{\partial U}^{n+\frac{1}{3}} \alpha_2^{n+\frac{1}{3}} \\ \frac{\partial S}{\partial U}^{n+\frac{1}{3}} \alpha_3^{n+\frac{1}{3}} \\ \frac{\partial S}{\partial U}^{n+1} \alpha_1^{n+1} \\ \frac{\partial S}{\partial U}^{n+1} \alpha_2^{n+1} \\ \frac{\partial S}{\partial U}^{n+1} \alpha_3^{n+1} \end{bmatrix}, \quad (3.26)
$$

which includes all necessary information to advance the solution by one single time step.

3.5 Intercell Flux Calculation

As mentioned in the previous section, the fluxes need to be found along the boundary at different time instances by solving Riemann problems between neighbouring cells' states, U. The Riemann Problem describes the discontinuous-boundary interaction between cells. The Riemann Problem is an initial-value problem for partial differential equations with initial condition

$$
U(x,0) = \begin{cases} U_L, & x > 0 \\ U_R, & x < 0 \end{cases}. \quad (3.27)
$$

The goal of the Riemann problem is to determine the state, U^*, on the boundary (x=0) in the future. Given the discontinuous nature of the Riemann problem at t=0, waves will always emerge from the discontinuity, and these waves separate states, U. In the context of the example provided by Figure 3.6, the desired state is state, U_L^*, where $\lambda_1, \lambda_2, \lambda_3$ are the wave speeds of the waves emerging from the discontinuity. For most simple cases, an exact solution to this problem exists, using the Rankine-Hugoniot jump conditions to describe the relation between the left and right states of a particular wave, λ. However, some PDEs have solutions to the

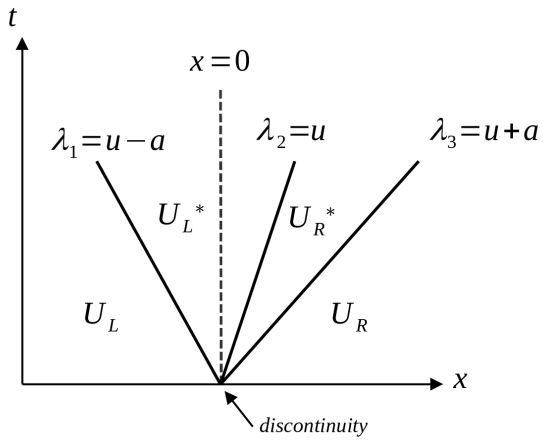

Figure 3.6: A diagram of the Riemann problem's wavespeeds and states

Riemann problems that cannot be found easily or cannot be written in closed form, which gives rise to the need for approximate Riemann solvers. Some of the approximate Riemann solvers used in this work include the Roe [Roe81] and HLLE [HLL83] [Ein88] approximate Riemann solvers.

Chapter 4

Implementation

4.1 Introduction

BLawB is a software developed for the efficient and accurate solution of hyperbolic balance laws using the discontinuous-Galerkin Hancock (DGH) scheme. Prior to this thesis, this software was only able to obtain numerical results when computing on a structured mesh. A structured mesh stores values such as node placement and cells in an ordered fashion, such as node 1 is next to node 2, and so on. This arrangement is intuitive, easier to deal with during development, and proves to be computationally faster, due to the organized nature of the storage of nodes and cells. However, complex geometries prove to be difficult when it comes to structured meshes. Often times, when dealing with a more complex mesh, a better solution is required around the parts that are more detailed, or more "odd", because stretching a structured mesh around these areas would compromise the mesh's integrity – this is to say solutions on this mesh would become less accurate. Unstructured meshes prove to be the solution to this problem, as they simply fill the required space with n-sided polygons (often triangles). This allows the polygons to be of regular proportion, avoiding a drop in accuracy. A comparison between a structured mesh and unstructured mesh is shown in Figure 4.1. In the context of this work, many files were developed in order to allow Unstructured Meshes to be used along with the DGH scheme. All header files detailed in this section were developed in this work.

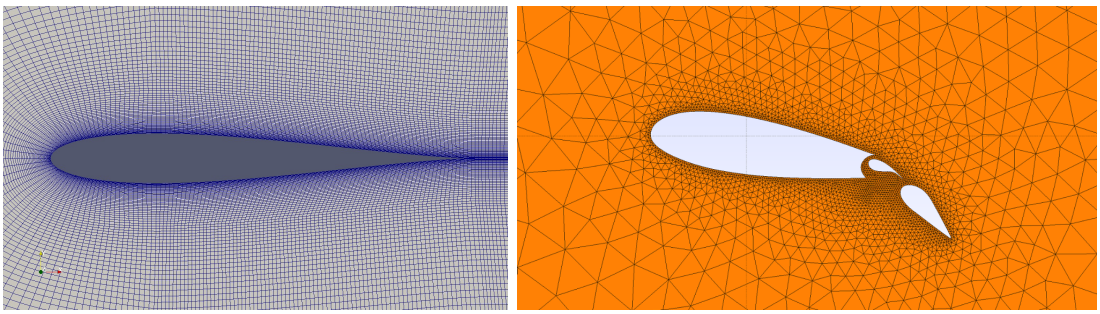

Figure 4.1: A comparison of a structured mesh (left) and an unstructured mesh (right)

4.2 Mesh Generation

The goal of unstructured mesh generation is to fill a defined geometry with n-sided polygons of desired size. In this work's case, triangles are used in 2D, and tetrahedra in 3D. This means generating nodes and cells. Unstructured mesh generation is relatively well-studied, and in consequence, many mesh-generation tools exist. The tool used in this work is GMSH, which is a meshing tool developed by Geuzaine and Remacle [GR09].

GMSH is a very versatile tool. It includes a GUI and a scripting language in which it is possible to create geometry to be meshed. GMSH can also accept CAD files if needed. Another very convenient feature which is extensively used in this thesis is the possibility to control the mesh's resolution in different parts of the geometry. It is therefore possible to smoothly transition between high resolution and low resolution. A great example of this is shown in Section 5.3.1, where a finer mesh is needed around the boundary of an airfoil to capture the boundary layer, and not needed anywhere else.

4.3 C++ Classes and Files

BLawB's Unstructured_Partition extension, entirely developed in this work, is comprised of many C++ classes, each with a different role. Unstructured_Partition_Base contains all the basic data that is contained in both 2D and 3D meshes, such as the nodes array, and cells array. It serves as a base class from which to create dimension-specific child classes. Unstructured_Partition_2D and Unstructured_Partition_3D are child classes to Unstructured_Partition_Base, and expand its scope to include dimension-specific functionality. The Unstructured_Partition_2D_Refine.h and Unstructured_Partition_3D_Refine.h files are used to complete refinement on 2D and 3D meshes, respectively. The Triangular_Element_Mapping.h and Tetrahedral_Element_Mapping.h files are used to integrate over triangles and tetrahedra using Gaussian quadrature.

The following is context that must be established in order to properly describe the implementation: After refinement, the mesh is divided into partitions, that can be distributed to numerous computational cores on a super computer. At any given time, a partition can have its resolution increased, and be subdivided into children, each with the same number of elements as the parent. Each child is assigned a unique number from each refinement event, and the list of these numbers, the refinement history, forms a signature that uniquely identifies each partition.

4.3.1 The Unstructured_Partition_Base Class

The Unstructured_Partition_Base class contains the fundamental components of a partition. It stores the partition's nodes and cells, which completely describe the mesh's geometry. A block signature is also stored within the base class, which is this partition's refinement history. It also, as the name suggests, serves as a base on which a 2D and 3D extension can be established. The class is structured to be able to efficiently read in input files, and establish and store basic information that describes a partition.

Storage Arrays

Name	Type	Size	Description
nodes	Node Type	Number of Nodes	The array of all the nodes' x, y and z locations included in the mesh
cells	Integer	Number of Cells	An array of cells, where each cell is an array pointing to the indices of the nodes of which it is comprised
block signature	Block Signature	Variable	The refinement history of this part of the mesh

Node Type: stores the physical x, y, and z coordinates. Handled by the Eigen library.

Block Signature: stores an array of refinement indices (0-3 in 2D, 0-7 in 3D).

Reading GMSH File

The Unstructured_Partition_Base class is initialized by reading in a GMSH file. The function first checks if the input file is a valid GMSH file, and then fills the arrays described in the previous section with the information it reads in. GMSH outputs a *.msh file that lists all necessary information to describe the mesh. To begin with, the node locations are listed (the x, y, and z coordinates of the nodes) in the $Nodes section. Following the nodes, the elements are listed in the $Elements section. Elements are entities that are comprised of one or a series of nodes. In the two-dimensional case, the $Element section will contain boundaries, comprised of 2 node indices plus a tag, and cells, comprised of 3 node indices plus a tag. Tags are special integer values that can be given to individual entities, and for this use case, tags put on boundaries are particularly important. In BLawB, specific boundaries are identified using tags, so that specific boundary conditions can be applied to them. These boundary conditions can be reflection, constant extrapolation, or fixed, for example.

4.3.2 The Unstructured_Partition Child Classes

The Unstructured_Partition child classes, or more specifically Unstructured_Partition_2D and Unstructured_Partition_3D, are the extensions to the Unstructured_Partition_Base class. They complete the base class' functionality and tackle everything that is dimension-specific. Amongst these dimension-specific functionalities are computing the mesh's faces, geometrical properties, and quadrature points. These child classes can be thought of as the main container for a single partition, as they contain all necessary information to describe it. Within the code, partition objects have the Unstructured_Partition_2D or Unstructured_Partition_3D types.

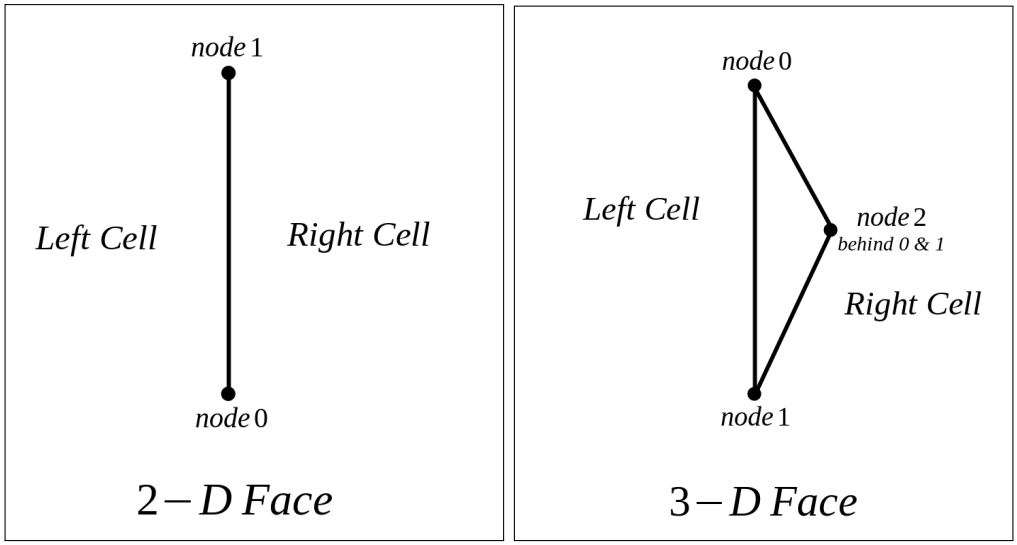

Figure 4.2: Diagram of 2D and 3D faces.

Storage Arrays

Name	Type	Size	Description
internal_faces	Face type	Number of internal faces	Array of all the faces which are bordering two cells.
boundary_faces	Face type	Number of boundary faces	Array of all the faces which are bordering only a single cell. They are part of the partition boundary.
volumes	Scalar	Number of Cells	Array containing each cell's volume.
centroids	Node type	Number of Cells	Array of each cell's centroid's x, y and z (if 3D) coordinates.
inertias	Matrix type	Number of Cells	Array of each cell's inertia matrix (2x2 for 2D, 3x3 for 3D).

Node Type: stores the physical x, y, and z coordinates. Handled by the Eigen library.

Face type: stores the face's 2 (2D) or 3 (3D) nodes, and the indices of the cells that are bordered by the face.

Computing Faces

After reading in the *.msh file, the *nodes* and *cells* vectors are filled in with the necessary information. For BLawB, computing an update in time to all cells values involves looping through all the mesh's faces, and computing the fluxes between the cells who are bordered by these faces. Hence the $internal\_faces$ and $boundary\_faces$ vectors must be filled. To do this, a

temporary faces vector is created, in which faces are added (groups of node indices), as well as a "left" and "right" cell. A convention is established based on the node-ordering to differentiate the sides. This convention is pictured in Figure 4.2.

Then, looping through the cells vector, the temporary faces vector is filled with every individual face (2 or 3 node pairing) each cell has, and the current cell number as the "left" cell. In two dimensions, there are cells comprised of 3 nodes, and these cells have three 2-node faces. In three dimensions, there are cells comprised of 4 nodes, and these cells have four 3-node faces. Given the fact that internal faces will always border 2 cells, it must be verified whether this specific face (2 or 3 node pairing) was already added to the temporary faces vector by a previous cell. If it was, it means the cell at which the index is in the loop, will be initially assumed to be the face's "right" cell.

At this point, the face has all information necessary to be fully defined, however the information is not necessarily stored correctly. For instance, at this point, the cells that are initially presumed to be the "left" and "right" cells may be swapped. To rectify this, a test is put in place to make sure the information respects the convention established in Figure 4.2. The test creates a vector from node0 to node1, as well as a vector from node0 to the third node of what is presently considered the right cell. The cross product is computed between the first and the second vector. If the result is positive, the convention is respected. If the result is negative, the right and left cell are swapped.

Once done looping through the cells vector, the temporary faces vector is filled with faces that have both a "left" and a "right" cell. All of these are added to the internal_faces vector. The temporary faces vector is also filled with faces that only have a "left" cell. These will be added to the boundary_faces vector.

4.3.3 The Triangular_Element_Mapping.h Class

As discussed in Chapter 3, The DGH scheme requires spacial integration across each cell and across their boundaries. In the current implementation, this integration is done using gaussian quadrature, within the Triangular_Element_Mapping.h class. This class is responsible for integration on triangular elements within a 2D mesh, and triangular faces within a 3D mesh. This class takes in the three nodes of which each triangular element is comprised, and computes a mapping able to translate any location within the computational reference domain to its physical counterpart. It can then, based on this mapping and its hard-coded quadrature rule, compute the triangle's centroid, area, unit normal, first moment of inertia, and mapped quadrature points. Each triangle's geometrical properties are found using

$$A = \sum_{i=1}^{n} w_i$$

$$c = \sum_{i=1}^{n} w_i p_i$$

Point	Weight	i	j
1	0.1099517	0.8168475	0.0915762
2	0.1099517	0.0915762	0.8168475
3	0.1099517	0.0915762	0.0915762
4	0.2233815	0.1081030	0.4459484
5	0.2233815	0.4459484	0.1081030
6	0.2233815	0.4459484	0.4459484

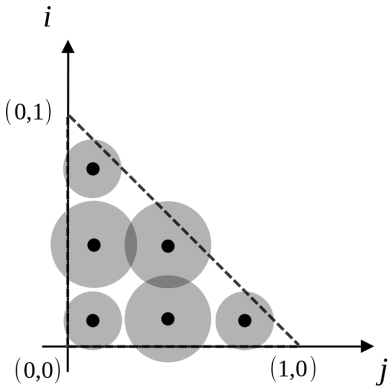

Figure 4.3: The reference triangle's quadrature point locations and weights.

$$Ixx = \sum_{i=1}^{n} w_i (p_{ix} - c_x)^2$$

$$Iyy = \sum_{i=1}^{n} w_i (p_{iy} - c_y)^2$$

$$Ixy = \sum_{i=1}^{n} w_i (p_{ix} - c_x)(p_{iy} - c_y)\,.$$

Before any time steps occur, the code uses this class to compute all of these values, and stores them within the main Unstructured_Partition class, to then be obtained from memory instead of being recomputed every time step.

As for the hard-coded quadrature rule, for the DGH scheme, exact results when integrating polynomials up to degree 4 are required. The quadrature rule depicted in Figure 4.3 does so. Its set of quadrature points and weights on the reference triangular element were obtained from [Burb]. Once the reference triangle's quadrature point locations and weights are found, each individual triangle on any given mesh can obtain its quadrature points by mapping them from the reference triangular element, as shown in Figure 4.4, using a bilinear mapping. This maps physical coordinates $[x, y]$ to computational coordinates $[\zeta, \eta]$,

$$x(\zeta, \eta) = (1 - \zeta - \eta)x_0 + \zeta x_1 + \eta x_2\,, \tag{4.1}$$

where x_0, x_1, and x_2 are the coordinates of the real-domain triangle to which the reference triangle is mapping. The matrix describing this coordinate transformation is defined as the Jacobian, $J = \frac{\partial x_i}{\partial \zeta_j}$,

$$J(\zeta_j) = \frac{\partial x}{\partial \zeta}\frac{\partial y}{\partial \eta} - \frac{\partial x}{\partial \eta}\frac{\partial y}{\partial \zeta}\,. \tag{4.2}$$

4.3.4 The Tetrahedral_Element_Mapping.h Class

In three dimensions, cells are tetrahedra, and faces are triangles. This section covers the Tetrahedral_Element_Mapping.h class and Gaussian integration on tetrahedral cells. The con-

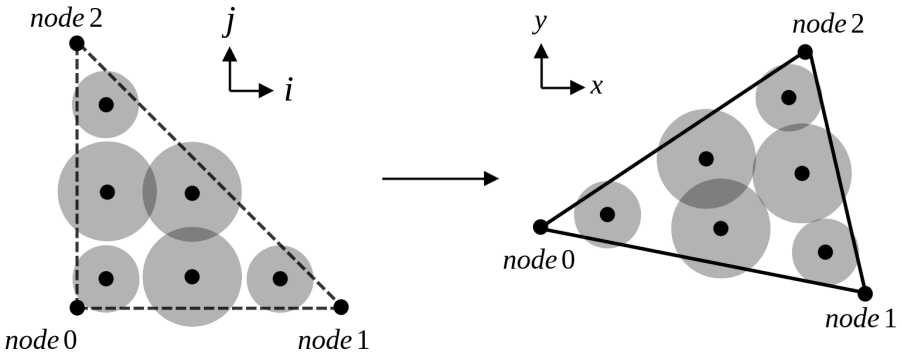

Figure 4.4: A bilinear mapping from the reference triangle to an arbitrary triangle with nodes 0, 1 and 2.

cept is extended from the triangle Gaussian integration in the last section. The Tetrahedral_Element_Mapping.h class computes the mapping using the tetrahedron's four nodes, and then uses its hard-coded quadrature rule to compute the tetrahedron's volume, centroid, three types of moments of inertia, and mapped quadrature points. Each tetrahedron's geometrical properties are found using

$$V = \sum_{i=1}^{n} w_i$$

$$c = \sum_{i=1}^{n} w_i p_i$$

$$Ixx = \sum_{i=1}^{n} w_i (p_{ix} - c_x)^2$$

$$Iyy = \sum_{i=1}^{n} w_i (p_{iy} - c_y)^2$$

$$Ixy = \sum_{i=1}^{n} w_i (p_{ix} - c_x)(p_{iy} - c_y)$$

$$Izz = \sum_{i=1}^{n} w_i (p_{iz} - c_z)^2$$

$$Izx = \sum_{i=1}^{n} w_i (p_{iz} - c_z)(p_{ix} - c_z)$$

$$Izy = \sum_{i=1}^{n} w_i (p_{iz} - c_z)(p_{iy} - c_z)\,.$$

All of these values for each tetrahedral element are computed and stored before time stepping, in order to avoid computing redundantly. First, a quadrature rule must be found that will work on a reference tetrahedron. The chosen set of quadrature points and weights on the reference

Point	Weight	i	j	k
1	0.218	0.568	0.144	0.144
2	0.218	0.144	0.144	0.144
3	0.218	0.144	0.144	0.568
4	0.218	0.144	0.568	0.144
5	0.021	0.0	0.5	0.5
6	0.021	0.5	0.0	0.5
7	0.021	0.5	0.5	0.0
9	0.021	0.0	0.5	0.0
10	0.021	0.0	0.0	0.5

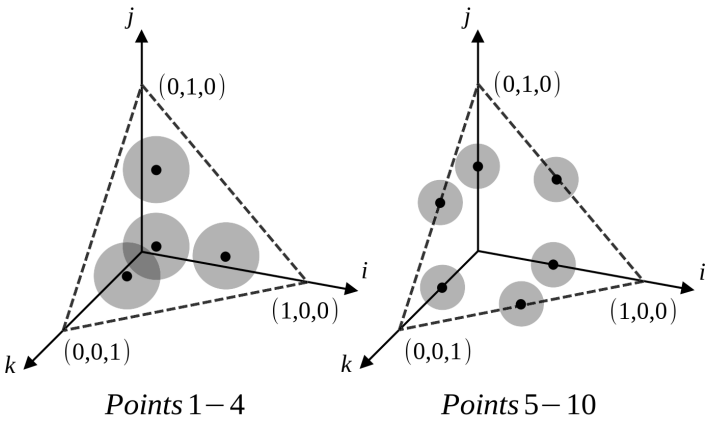

Figure 4.5: The reference triangle's quadrature point locations and weights. On the right, the circles around the points represent the point's associated weight.

tetrahedral element were obtained from [Bura] and are shown in Figure 4.5. The quadrature points on any arbitrary tetrahedral can be mapped from this reference tetrahedral using a trilinear mapping.

4.3.5 The Unstructured_Partition_Refine Classes

The Unstructured_Partition_Refine_2D.h and Unstructured_Partition_Refine_3D.h files handle refinement. They are called by the main part of the code whenever a partition is deemed in need of refinement, based on user-defined criteria. When refinement takes place in the Unstructured_Partition_Refine classes, two main things happen to the partition. First of all, each cell is split to increase the mesh's resolution. Then, the totality of the mesh gets divided into partitions, or "blocks" in the context of this implementation. While running this code on distributed clusters, these blocks have the opportunity to be distributed to other processors to work in parallel.

2D Sub-Splitting

This part of the two-dimensional refinement process splits a 2D cell into 4 smaller, similarly sized triangles, as demonstrated in Figure 4.8. This procedure adds a node at the midpoints of each segment of the original triangle, and creates four smaller triangles, as shown. As can be seen from Figure 4.7, a significant amount of resolution is added after performing this process once on the whole domain. This effectively quadruples the number of cells and nearly doubles the number of nodes in the mesh.

3D Sub-Splitting

This part of the three-dimensional refinement process takes each tetrahedron cell and splits it into 8 sub-cells, as represented in Figure 4.9. This procedure takes the midpoints of each

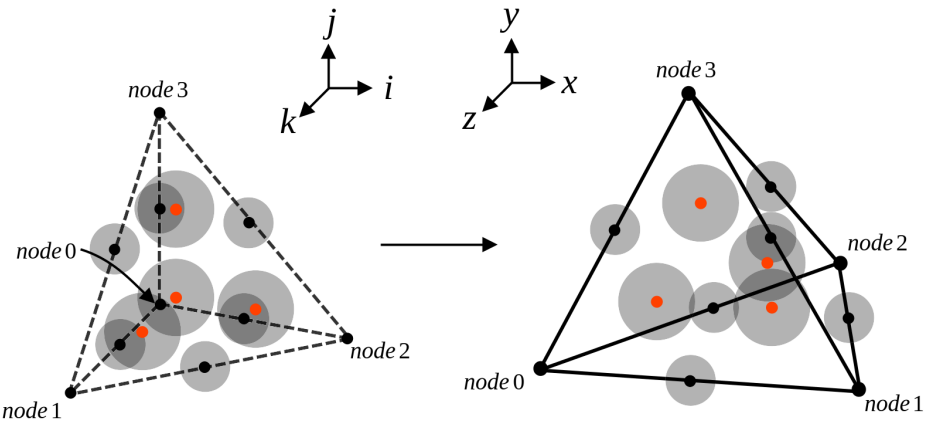

Figure 4.6: A trilinear mapping from the reference tetrahedron to an arbitrary tetrahedron with nodes 0, 1, 2 and 3. The red points are the first four quadrature points, and are located within the tetrahedron's volume. The rest are on the edges. The circles around the points represent the point's associated weight.

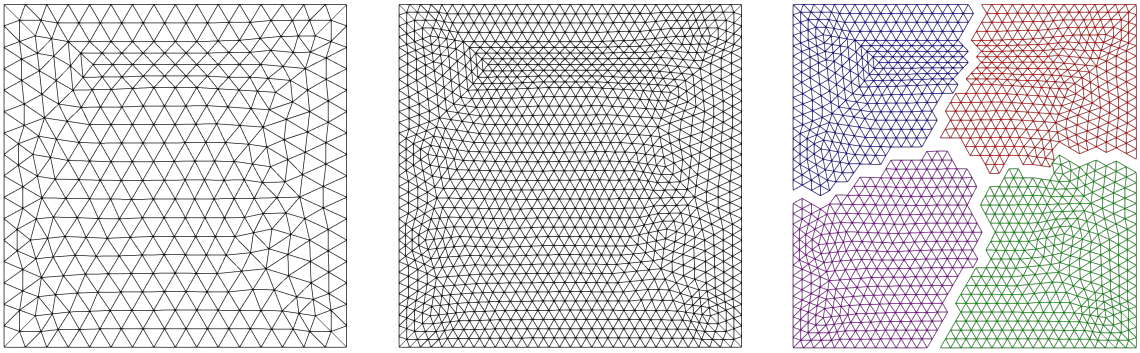

Figure 4.7: A square two-dimensional Unstructured Partition going through one level of refinement. From left to right, the parent block, the refined block, and finally the partitioned blocks. The partitioned blocks have been spaced out to illustrate the separation.

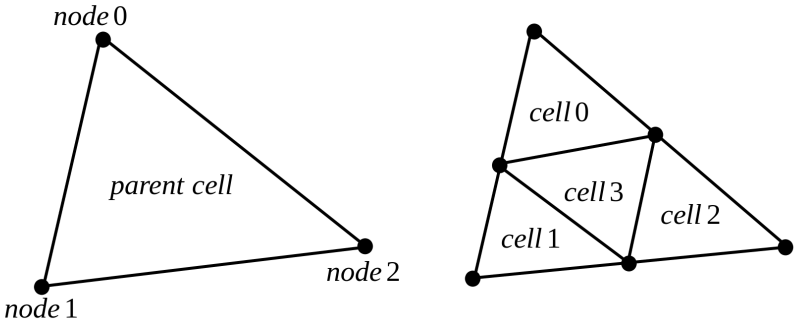

Figure 4.8: A diagram demonstrating the sub-splitting process in two dimensions, with the parent cell on the left, and the four child cells on the right.

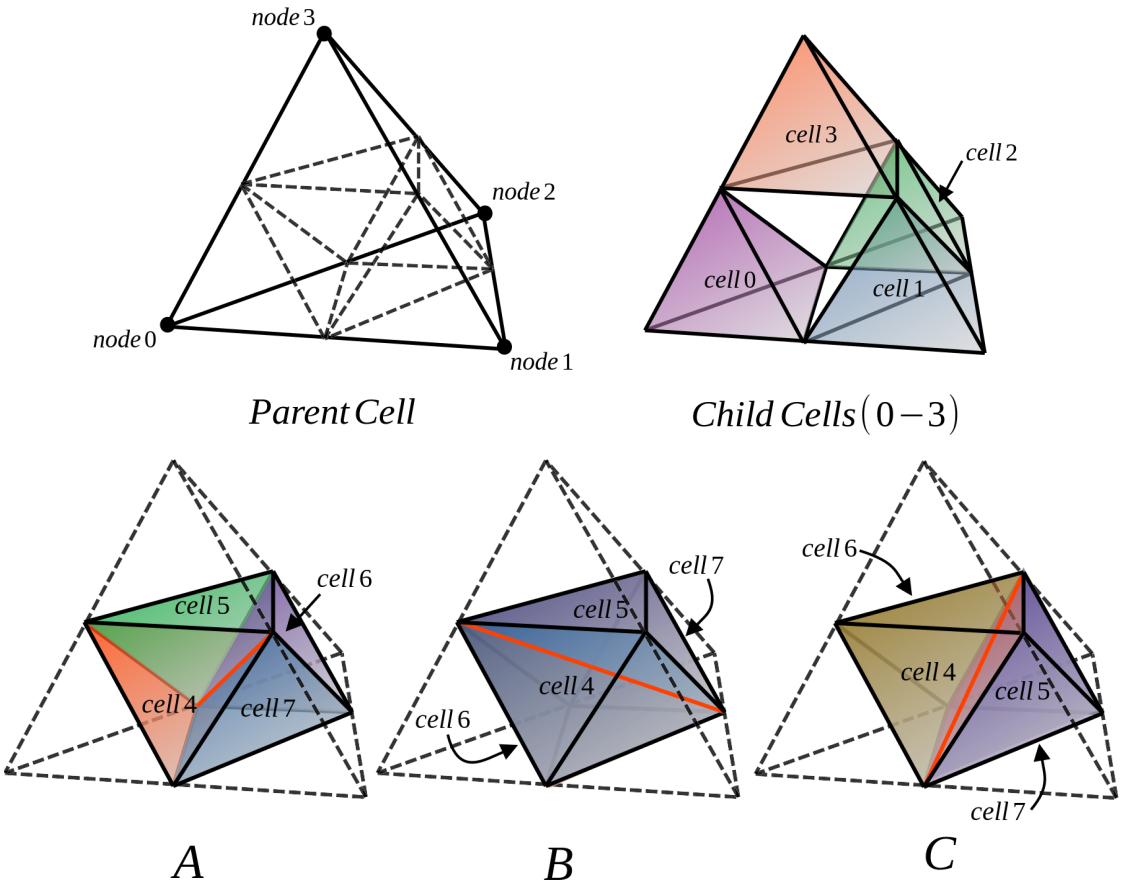

Figure 4.9: A diagram demonstrating the sub-splitting process in three dimensions, with the parent cell on the top left. The top right figure shows child cells 0 to 3, and on the bottom, the 3 possible ways to split the remaining space into the remaining 4 child cells, in order to maximize the child cells' quality.

segment connecting two nodes and splits each face of the tetrahedral into four. The first four child cells are as depicted in the top right of the figure in question, and the remaining 4 being in the center. However, there are 3 total ways to split this center volume into 4 tetrahedrals. The chosen method is the one that will prevent creating irregularly thin tetrahedra. To achieve this, the method that minimises the length of the edge connecting all four tetrahedra is selected, which is the edge depicted in red in Figure 4.9.

Partitioning

This part of refinement takes the modified high-resolution partition that was created in the previous sub-splitting step and divides it into blocks. In two dimensions, the partition divides into 4 blocks, and in three dimensions, 8 blocks. The goal is to distribute the partitioned blocks to different processors for parallel computation. The time it takes to complete an iteration is directly depend on the processor containing the largest block, because at every iteration, each individual cell must be updated. The processor containing the largest number of elements will take the longest to compute. For this reason, it is essential that blocks remain of similar total amount of elements during this partitioning step. This will be done by a software package called METIS.

METIS

METIS is a software package for partitioning graphs, partitioning meshes, and producing low-fill orderings of sparse matrices. It is developed by Karypis and Kumar [KK99]. The particular functionality of interest for this study is mesh partitioning. Figure 4.7 shows a mesh that has been partitioned into 4 blocks by the METIS_PartMeshDual routine. This routine takes each cell of the mesh as a vertex in a graph, the totality of which is coarsened until the graph is only a few hundred vertices. At this point the coarser graph goes through a partitioning algorithm which minimizes the number of edges straddling different partitions. This is often referred to as the *edge cut*. The graph is then uncoarsened, and the partitioning is projected onto the now finer graph. As Figure 4.7 shows, not only does the process minimize the edge cut, but also creates partitions of similar size to balance the computational load, if these partitions get assigned to different processors. This process is fast with large meshes due to the partitioning taking place on a smaller, coarsened representation of the initial mesh. This can create slightly less efficient partition separations, but definitely proves to be worth the sacrifice when weighing in the time savings. METIS does not see any difference between 2D and 3D meshes since it converts the mesh to a connectivity graph, which then gets split into any specified amount of parts, which makes it suitable for two-dimensional and three-dimensional refinement alike. A representation of METIS' purpose is shown in 3D in Figure 4.10.

Completing the refinement

Once the partitioning is complete, METIS_PartMeshDual outputs two arrays. The first is an array that stores the specific block to which each individual cell has been associated. The

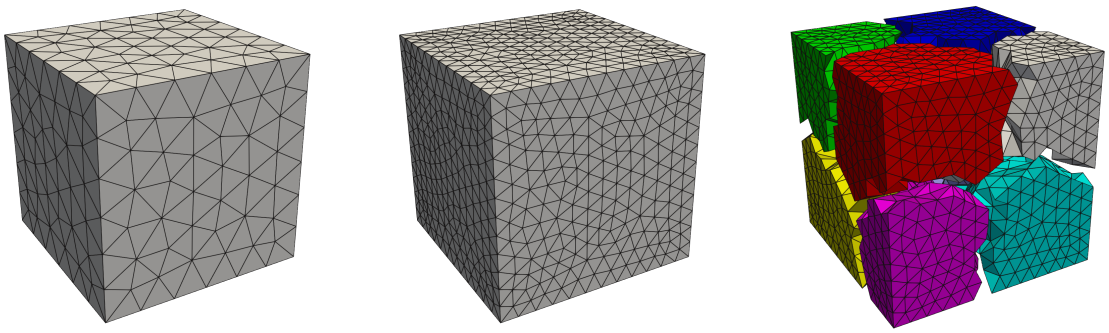

Figure 4.10: A cubic three-dimensional Unstructured Partition going through one level of refinement. From left to right: the parent block, the refined block, and finally the partitioned blocks. The partitioned blocks have been spaced out to illustrate the separation.

second array does the same for the nodes. Using these arrays, new blocks are created using their associated cells and nodes contained within the original block. Cell and node geometry get passed to the children blocks, and so does the solution within the cells. Each of these blocks then go through their individual initializations, including computing their internal faces, their cells' geometrical properties, and boundaries. Following this, METIS' output alongside the newly created blocks are used to compute a 3rd array, a list of boundary index and block index pairs storing which faces on which blocks are connected. The use of this new array is further described in Section 4.4, where this connection is concretized within each block, in order for boundary cells to be able to compute their update.

4.4 Block Connectivity & Parallel Computing

Communication between cells is crucial to CFD. Many CFD methods require complex stencils, which are a map of communication that dictate which group of cells a particular cell depends on to compute its update. In the DGH method's case, this stencil is optimally small. Only communication with immediately neighbouring cells is necessary. This simplifies refinement quite a bit, given that a cell that finds itself on a block's boundary only requires knowing which single cell on a neighbouring block it must communicate with. To create this inter-block communication, an array of each block's boundary connectivity must be constantly maintained, and updated as refinement takes place.

Refinement can be triggered at any point during a calculation, by any particular block of the mesh. When refinement takes place, the parent block splits itself into 4 new blocks in 2D, and 8 new blocks in 3D. These new blocks have new cells and new nodes, which were created during refinement, but the new blocks also have new boundaries, which need special attention. Some of these are boundaries that previously existed on the parent block, and some of these boundaries are completely new. Boundaries that existed on the parent block get passed down from the parent, whether they were boundaries of the whole domain (Domain Boundaries), or were boundaries that connected that block to a neighbouring one (Interblock Boundaries).

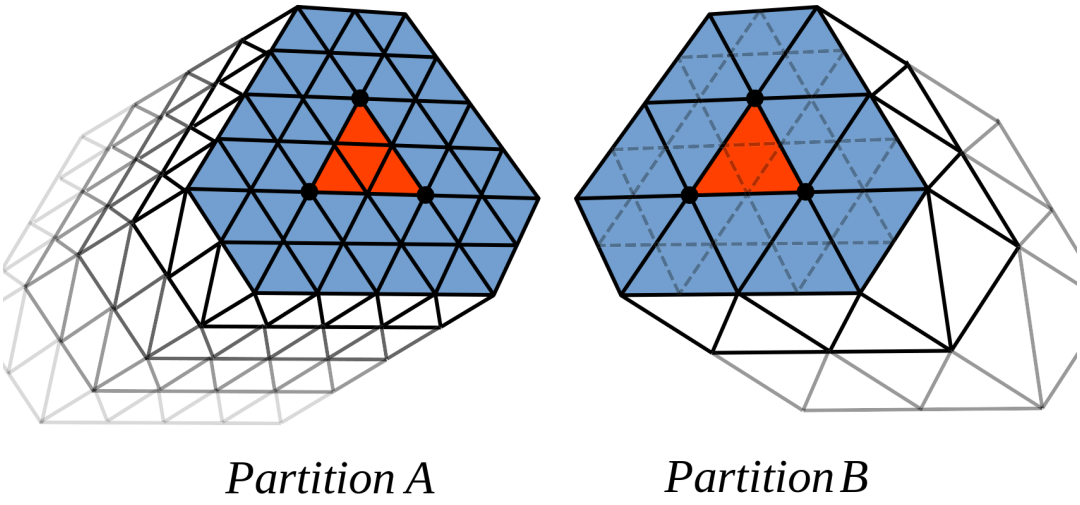

Figure 4.11: A diagram showing the differing resolutions from partition A to partition B, and their connectivity. The coloured faces are interblock boundaries linking faces between the two partitions.

The Interblock Boundaries that were not there before however, must be created, and added to the connectivity. This establishes the communication between the newly created partitions, and allows the whole mesh to continue computing updates to the solution. A list of these new boundaries is computed immediately after the partitioning process, and is described in Section 4.3.5.

For example, in the blue partition in Figure 4.7, the boundaries on its left and top edge would have been defined before refinement, and passed down from its parent. These could have been domain boundaries, if the parent block had no neighbour, or interblock boundaries if the parent block was originally connected to neighbouring blocks. The blue partition's remaining boundaries (on the lower edge and right edge) need to be created, and properly linked to their neighbours – the bottom edge to the purple block, and the right edge the red block.

Once the connectivity of all blocks is established, each block can be put on different processors. All blocks can begin to compute their internal cell's updates at the same time given that all information required to do so is local. When it is time to compute their boundary cells updates, the blocks use the Message Passing Interface (MPI) to communicate the necessary information. They know which cells on other processors are immediately neighbouring each of their boundary cells, and fetch the necessary information through MPI, while sending their own boundary cell information so blocks located on other processors may go through the same process.

4.4.1 Composite Boundaries

Boundaries, in a block-connectivity context, are called composite boundaries. A composite boundary is a container that can contain either a domain boundary, or an array of 1 or more

interblock boundaries. A composite boundary is associated to each boundary face of each partition. Looking at Figure 4.11, two connected pieces of a domain are noticed, where the blue region denotes their connected faces. In this context, these blue faces are composite boundaries containing interblock boundaries. On Partition B, the resolution on the face is higher than that of its cells. This boundary splitting happens to allow the connection with Partition A. Composite boundaries in the blue region of Partition B must split themselves into four to allow them to match up with the composite boundaries on Partition A. In this situation, a composite boundary holds a vector of 4 interblock boundaries.

4.4.2 Transformations

A triangular boundary connecting one partition to its neighbour contains the same nodes on one partition as on the next, but, in almost all cases, they will not be stored in the same order, due to the way a block computes its boundaries during the block's initialization. For this reason, when interblock boundaries are created, a transformation is computed, storing the information describing the difference between the neighbouring blocks' connected faces. In 3 dimensions, boundaries are triangles, hence the transformation uses the node locations on each partition to find out how many times one partition's nodes must be rotated, and whether or not the order must be reversed, in order to match a partition's face with its corresponding face on the other partition. In 2D, boundaries are simply lines, so transformations simply compute whether or not the node order must be reversed.

In the context of Figure 4.11, the transformation concept gets used when Partition B's face must split into four. In order to match up with the correct corresponding faces on Partition A, Partition B's faces get associated to their neighbour's faces according to the transformation that was computed when the connection between the two partitions was first conceived. There are additional situations where the transformation information is necessary. As mentioned previously, a 6-point quadrature rule is used to integrate over the surface of triangles. These 6 points are placed in specific locations, based on a triangle's node locations. In 3D, these quadrature points are found on boundaries that connect two partitions, so once the quadrature points are computed, their order is adjusted according to the boundary's transformation, so they match up with the corresponding quadrature points on the neighbouring partition.

4.4.3 Block and Boundary Signatures

It can be easily seen how communication between refined partitions becomes an increasingly difficult task as refinement is done dynamically in multiple layers. This is why a system is established within this code to keep track of refinement. To begin with, the Block_Signature concept is created. Every block has a Block_Signature, and its goal is to keep track of every step of refinement that this particular block has been through in the past, and which number partition it was during each refinement step (partition 0 to 3 in 2D, and 0 to 7 in 3D). Every new refinement event is appended to the refinement history.

Alongside this, the Boundary_Condition_Signature.h class is established. Any particular

boundary within a block will have a composite boundary, as was established previously. If this boundary is not at the edge of the whole mesh's domain, its composite boundary will contain one or more interblock boundaries, which each has a Boundary_Condition_Signature. This signature stores two Block_Signature objects, corresponding to the blocks on either side of the interface, and two indices. The two indices correspond to the index used within the neighbouring blocks to store the current boundary. This is all the information needed to uniquely identify a particular interblock boundary.

Every time refinement takes place, the blocks that are involved update their Block_Signature, and since all involved boundaries know where to find their neighbour, they can communicate with them and update their neighbours Block_Signature and boundary index, as well as their own. This ensures that, at all times, all boundaries know to which boundary on which block they are associated. This is essential for continuing iterations, and performing more refinement on any particular partition if necessary.

4.4.4 Block Communication

Communication becomes one step more complex when blocks are located on different MPI processes. Managing message-passing between blocks located on different CPUs is done with what is referred to as the communication hub. Each MPI process has one of these objects, and all information that must be sent to different MPI processes is done through it. This includes solution data in neighbouring cells, refinement data of neighbouring blocks, as well as CPU rank data for load-balancing.

Each communication hub on each MPI process has a list of send/recieve buffers, which can be imagined as a data tunnel connecting two MPI processes, and is associated with a specific pair of boundaries, each on a different block. This list must be dynamically modified as the code runs, given that blocks located within a given MPI process can be moved or refined. Whenever this happens, the list of send/recieve buffers is modified to include new interblock boundaries, by passing both its boundary signature, and neighbouring CPU rank. When the list gets modified, memory gets reallocated for solution data, refinement information, and CPU information of the new list. The interblock boundaries link their send and receive pointers with their associated segment within their MPI process' buffer. The connection hub is simply a framework that allows connected faces to point to each other's data's location within memory.

MPI messages are sent early within the time-update process using non-blocking MPI calls, allowing the messages to be received throughout the network, to ensure all MPI processes within the network have received all required information by the time they have finished doing all computations that can be done locally. Two exchanges of information per interblock connection need to be sent/received every time step, to communicate solution data and slope limiting data. These two pieces of information per interblock connection are pooled into buffers, and sent as a single MPI message per time step to MPI ranks that have neighbouring boundaries. An additional message must be passed globally in order for all processes to agree on a time step size.

Chapter 5

Numerical Results

In this section, results obtained by the code are shown. To begin with, an implementation analysis is displayed, demonstrating this code's scalability and accuracy. Then, the Euler equations are detailed, and four cases solving these equations are shown. A bump case showcases the code's adaptive refinement, a sound diffusion case shows unstructured meshes' flexibility in representing complex geometries, a case demonstrating a Kelvin-Helmholtz instability case shows the code's ability to run large problems on large parallel computers, and a 3D wing case shows the code's ability to perform 3D calculations. Following this, the ten-moment model and its equations is described, and a multi-element airfoil case is shown solving these equations.

Most calculations in this section were run on the Cedar heterogeneous cluster, located at Simon Fraser University, Vancouver, British Columbia. Cedar has a total of 94,528 CPU cores, comprised of Broadwell, Skylake, and Cascade lake cores. The cluster also has 1352 GPU devices, though GPUs are not used in this work. Cedar was sold and is supported by Scalar Decisions, Inc. Its nodes were manufactured by Dell, and the interconnect is from Intel. Cedar is named after the Western Red Cedar, British Columbia's official tree.

5.1 Implementation Analysis

This section demonstrates the efficiency and accuracy of BLawB when it is used in conjunction with unstructured meshes. First, the order of accuracy of the DGH scheme on unstructured partitions is studied. This is done by doing a convergence study on both the 2D and 3D implementations. Next, BLawB's scalability is tested by solving large problems using unstructured meshes on large clusters and performing a strong scalability study.

5.1.1 Convergence Studies

This work's convergence study takes a simple linear convection-relaxation problem for which the exact solution is known. This problem is then run using increasing mesh resolution, to observe the mesh resolution's effect on the accuracy of the computed solution. The linear

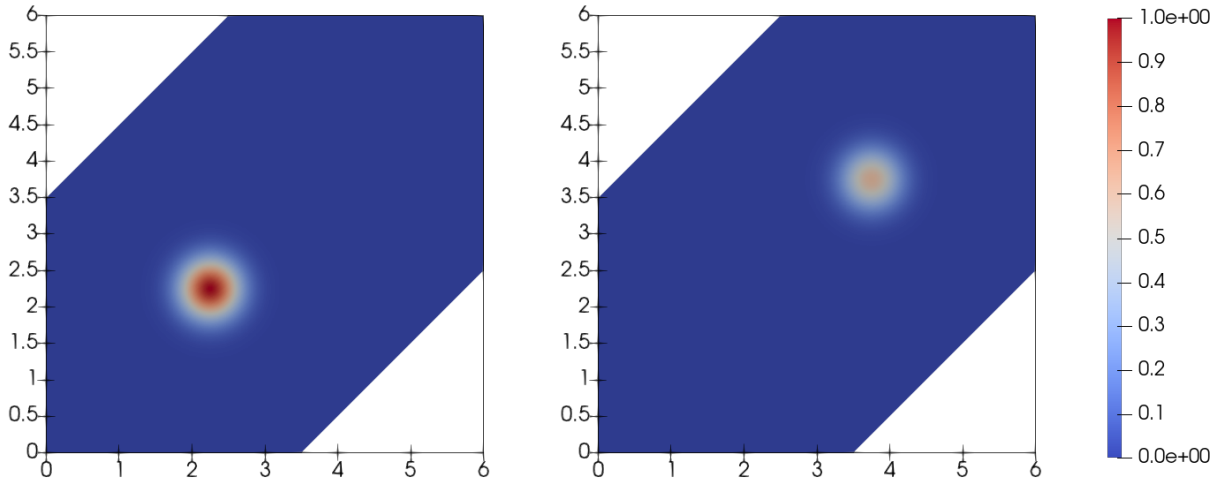

Figure 5.1: Initial and final states of the two dimensional convergence study, exact solution.

convection-relaxation equation can be defined as

$$\frac{\partial U}{\partial t} + u_i \frac{\partial U}{\partial x_i} = -\frac{U}{\tau}\,, \tag{5.1}$$

where u_i is the speed of convection, and τ is the relaxation factor. The exact solution to this equation is known to be

$$U(x_i, t) = U_0(x_i - u_i t)e^{-\frac{t}{\tau}}\,, \tag{5.2}$$

where U_0 is the function describing the initial condition. The error can be obtained by comparing the computed results to the exact results using the formula

$$l^2_{Error} = \sqrt{\sum_{i=0}^{N} V_i (U_{ie} - U_i)^2}\,, \tag{5.3}$$

where V_i is cell i's volume, U_{ie} and U_i are the values of the exact solution and computed solution at cell i respectively. Once the error is obtained for meshes with increasing resolution, the scheme's order is found using

$$O_k = \frac{\ln \frac{E_{k-1}}{E_k}}{\ln \frac{N_k}{N_{k-1}}}\,, \tag{5.4}$$

where E_k and N_k are the error and number of elements per dimension respectively at resolution k.

2D Convergence Study

In two dimensions, the domain for the convergence study consisted of a square with two chamfered corners, shown in Figure 5.1. The initial condition is a Gaussian bump centered at

Table 5.1: Results for the two-dimensional Convergence study.

n	Elements$^{\frac{1}{2}}$	l^2_{Error}	**Order**
1	38.78	9.46×10^{-3}	-
2	77.56	1.62×10^{-3}	2.54
3	155.13	2.86×10^{-4}	2.50
4	310.25	6.13×10^{-5}	2.22
5	620.50	1.47×10^{-5}	2.06
6	1241.01	3.65×10^{-6}	2.01

$(2.25, 2.25)$, defined as

$$U(x_i, 0) = e^{-k((x-x_c)^2+(y-y_c)^2)} \qquad x_c = y_c = 2.25 \qquad k = 6\,, \tag{5.5}$$

where k is the bump coefficient, and (x_c, y_c) are the bump coordinates. The speed u_i given to the convection equation is $u_x = u_y = 1$, and the relaxation factor is chosen to be $\tau = 1$. The solution is time marched to a time of $t = 0.5$. Given the speed vector and relaxation factor, the Gaussian bump can be expected to lower in intensity, and move up and to the right. Both the initial condition and exact solution are plotted in Figure 5.1. This mesh is specially created to ensure enough space for the boundaries to have no effect on the evolution, while minimizing the inclusion of cells that have no effect on the computation. This is to say that the mesh ensures that values on the border remain below machine zero throughout the computation. For this problem, the HLL flux function was used, and no slope limiter was used.

As can be seen in Table 5.1, the convergence obtained was not the one expected. The results show that the scheme displays 2nd-order accuracy on the unstructured triangular mesh, while 3rd order was expected. Yoshifumi Suzuki confirms that the scheme is 3rd order accurate on quadrilateral, structured, meshes [Suz08]. Suzuki mentions that there doesn't seem to be an apparent reason as to why it would not be 3rd order accurate on triangles, but does not confirm or prove anything. Although a bug in the code can never be ruled out, quadrature points, geometrical quantities, and the quadrature rule have all been tested extensively and are outputting expected values. As far as can be told based off of this study, it seems the DGH scheme may be only 2nd order accurate on triangular meshes.

3D Convergence Study

In three dimensions, the domain for the convergence study consists of a cylinder angled in the $(1, 1, 1)$ direction. The initial condition is a Gaussian bump centered at $(1.5, 1.5, 1.5)$ defined as

$$U(x_i, 0) = e^{-k((x-x_c)^2+(y-y_c)^2)+(z-z_c)^2)} \qquad x_c = y_c = z_c = 1.5 \qquad k = 9\,, \tag{5.6}$$

Table 5.2: Results for the three-dimensional Convergence study.

k	Elements per Dimension	l^2Error	Order
1	16.21	3.73×10^{-2}	-
2	30.50	1.45×10^{-2}	1.49
3	61.20	3.36×10^{-3}	2.10
4	121.68	7.34×10^{-4}	2.22
5	242.03	1.74×10^{-4}	2.09

where k is the bump coefficient. The speed u_i given to the convection equation is $u_x = u_y = u_z = 1$ and the relaxation factor is $\tau = 1$. The solution is time marched to a time of $t = 0.5$. Just like in two dimensions, the mesh was created to prevent the boundaries from influencing the computation. The boundary of the domain remains at machine zero throughout the computation. As it was in two dimensions, the HLL flux function was used, and no slope limiter was used. As is the case for the two-dimensional convergence study, 2nd-order accuracy emerges as the resolution of the problem is increased.

5.1.2 Scalability Study

It is previously mentioned that one of the DGH scheme's main advantages is its high parallel efficiency. To verify our implementation's efficacy at running on large computers in parallel, a strong scaling study was conducted. In a strong scaling study, the problem size is kept constant, and the number of cores is increased. Two metrics are obtained to illustrate the parallel performance of the implementation, Speed-Up, S, and Efficiency, E. Speed-up is the factor by which the speed has increased when additional cores have been made available

$$S = \frac{t_1}{t_n}, \tag{5.7}$$

where t_1 is the wall-time of the computation on the smallest number of cores, and t_n is the wall-time of the computation on subsequent, bigger-cluster. In this case, t_1 is the wall-time of the computation on 48 cores, and t_n is the wall-time of the computation when doubling this number, i.e. 48, 96, 192, 384 and 768 cores. For perfect scaling, the Speed-up factor should be equal to the core increase factor. Efficiency is how close the implementation is to this perfect scaling, and is defined as

$$E = \frac{S}{n}. \tag{5.8}$$

To perform the scalability study, the Euler equations from Section 5.2 are used, and the Kevin Helmholtz problem described in Section 5.2.3. Increasing numbers of cores are made available between different runs, starting at 48 cores, and doubling every time. As can be seen in Figure 5.2, this implementation scales nearly linearly, with efficiency dropping to only about 86 percent when using 768 cores in parallel.

# CPUs	Time (s)	Speed-up	Efficiency
48	34638.55	1	1
96	17651.58	1.96	0.98
192	9160.05	3.78	0.94
384	4782.58	7.24	0.91
768	2512.34	13.78	0.86

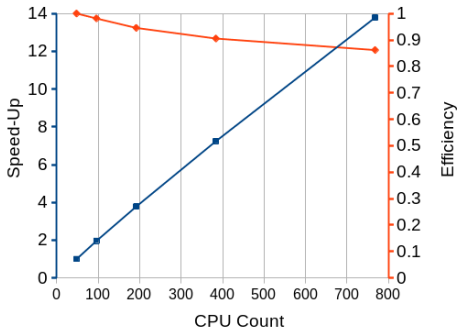

Figure 5.2: Results for the strong scalability study on the Kelvin-Helmholtz Instability problem, from 48 CPUs to 768 CPUs.

5.2 Euler Equations

In this section, the Euler equations are detailed and then solved with this work's code. The Euler equations are first solved in a Bump case, in which this code's adaptive mesh refinement's advantages are demonstrated. Following this, the Euler equations are solved in a Sound Diffusion case, in which the code's adaptability to odd geometries is shown. Finally, the Euler equations are solved in a Kelvin-Helmholtz case, in which the benefits of being able to run on large computers are shown.

Formulation

The compressible Euler equations take the form

$$\frac{\partial U}{\partial t} + \frac{\partial F_i}{\partial x_i} = 0\,, \tag{5.9}$$

where

$$U = \begin{bmatrix} \rho \\ \rho u_i \\ \frac{p}{\gamma - 1} + \frac{1}{2}\rho u_i u_i \end{bmatrix}, \quad F_i = \begin{bmatrix} \rho u_i \\ \rho u_i u_j + p\delta_{ij} \\ u_j(\frac{\gamma p}{\gamma - 1} + \frac{1}{2}\rho u_i u_i) \end{bmatrix}. \tag{5.10}$$

The primitive variables, stored within the primitive solution vector W, are

$$W = \begin{bmatrix} \rho \\ u_i \\ p \end{bmatrix}. \tag{5.11}$$

The Euler equations describe inviscid, adiabatic, compressive gas flows. They equate to the Navier-Stokes equations with zero viscosity and zero thermal conductivity.

5.2.1 Bump Case

The first case is a classic bump case with Mach 2 supersonic flow. This case has high variability when it comes to the mesh resolution required to properly capture the correct solution,

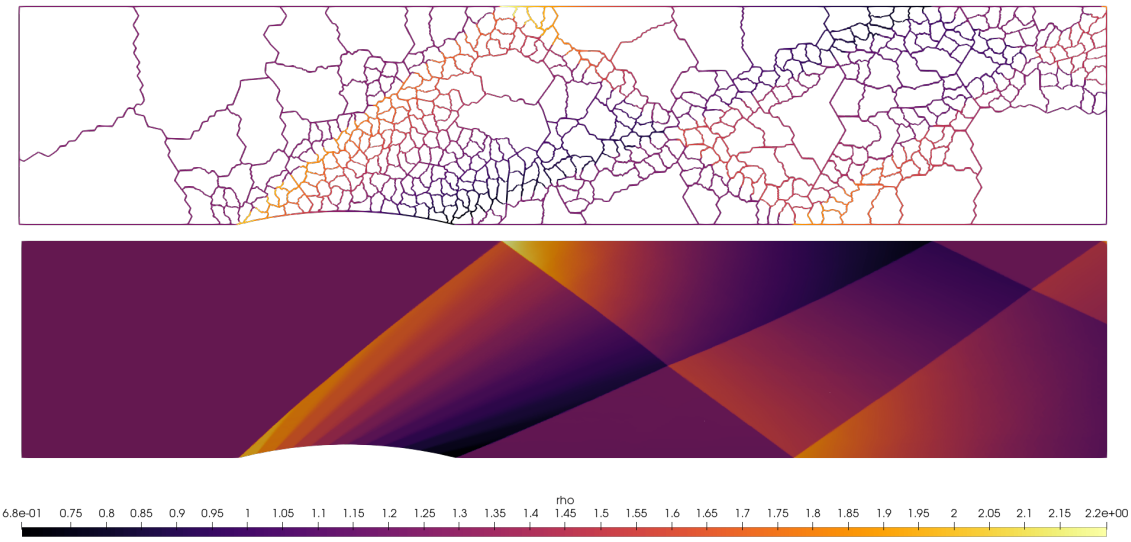

Figure 5.3: Results of the Bump case. The top shows all the blocks of the mesh, visualizing its refinement. Each block contains approximately 1200 cells.

and for this reason displays the code's adaptive mesh refinement (AMR) well. The flow was done over $x = (-1, 4)$ and $y = (0, 1)$, with a bump located on the bottom boundary between $x = 1$ and $x = 2$. The bump is a circular arc, centered at $(1.5, -2)$. The domain's initial condition is set uniformly at the flow conditions

$$\rho = 1.225 \text{ kg m}^{-3},$$

$$p = 101325.0 \text{ Pa},$$

$$u_x = 680.58 \text{ m s}^{-1},$$

$$u_y = 0.0 \text{ m s}^{-1}.$$

The $x = -1$ boundary is held fixed throughout the computation at these flow conditions as well. A reflection condition was applied to the top and bottom boundaries, and a zero-gradient condition was applied on the remaining $x = 4$ boundary. The solution was time-stepped using a CFL number of 0.3, HLLE flux function, and Venkatakrishnan slope-limiting, to a final time of $t = 0.02$. The final solution is shown in Figure 5.3. AMR was set to be triggered every 1000 iterations, and AMR triggered refinement in a block based on the l^2-*norm* of the gradient of the density within that block. Any block containing a l^2-*norm* of the gradient that was higher than 40kg/m^4 would be refined, up to a maximum of 5 levels of refinement. With the initial mesh containing approximately 1200 cells, each refined block will also have approximately 1200 cells. A progression of the mesh's refinement is shown in Figure 5.4.

As can be seen in Figure 5.4, the adaptive refinement gives the mesh the resolution necessary to properly capture the sharpness of the shock wave. Areas that do not need that much resolution, such as to the left of the bump or even between the shock waves, are kept at lower

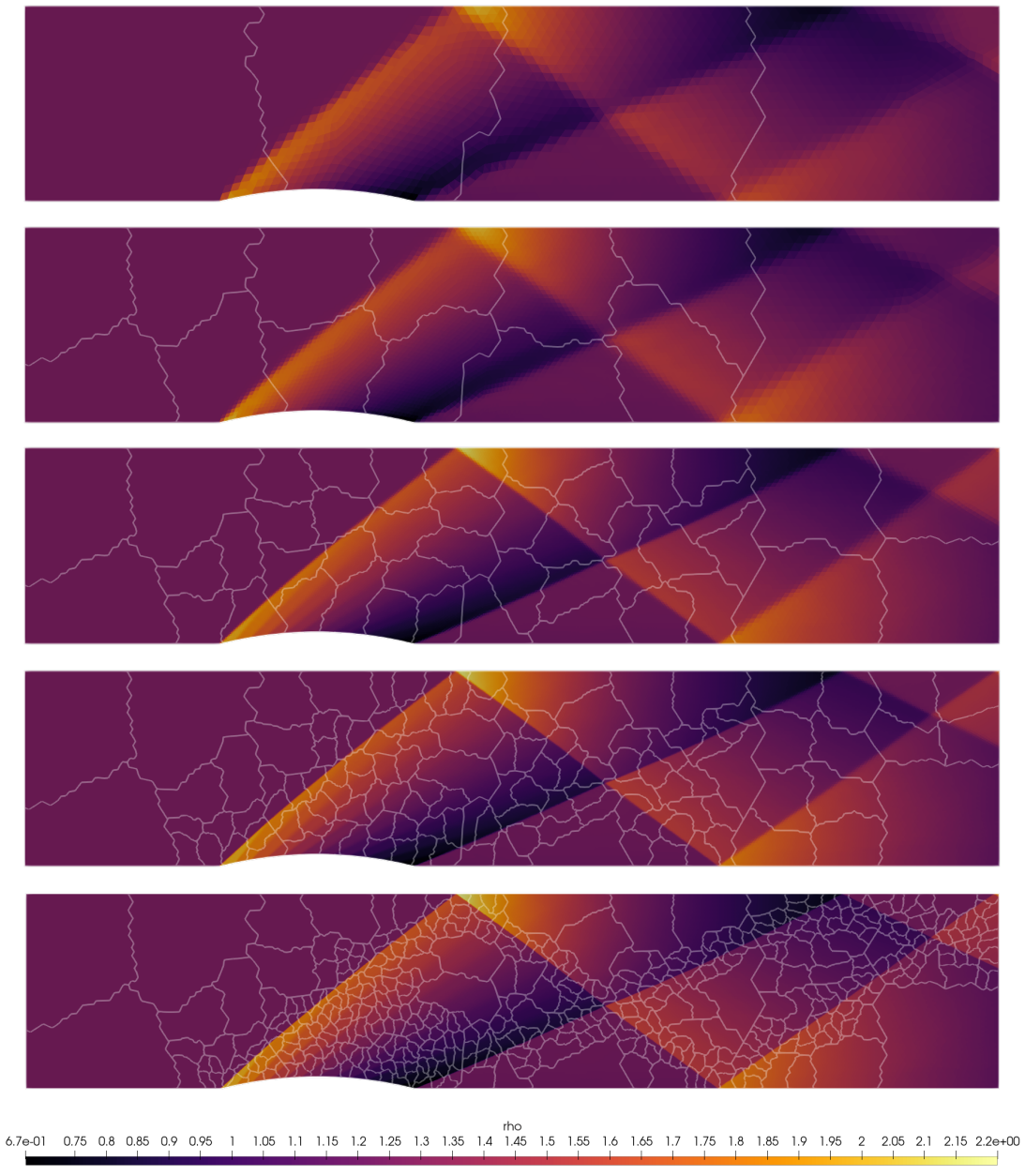

Figure 5.4: Results of the Bump case. Each refinement step is shown, from coarsest (top) to most refined (bottom). A global limit of 5 refinement levels was set before the calculation. From top to bottom, the cell totals are 5167, 16804, 55465, 156205, and 569167.

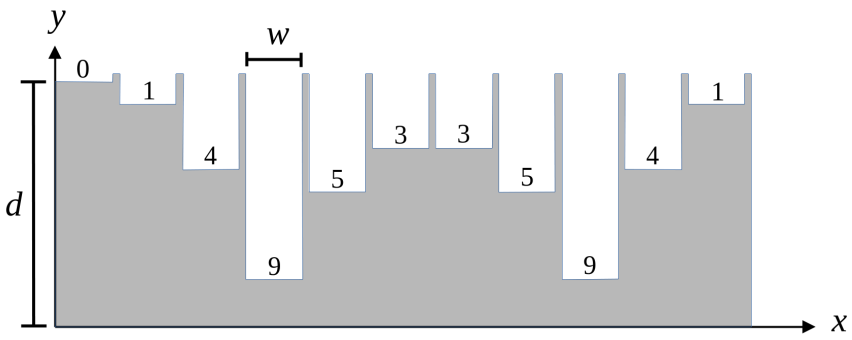

Figure 5.5: This is an N11 QRD, a type of sound diffusion panel profile. Its well-depth pattern is shown, from well 1 to 11. d is the design depth, w is the well width, and all remaining numbers are the depth of the wells, defined as multiples of $\frac{d}{11}$, the depth unit.

resolution to minimise runtime. This flexibility is especially useful when it is not known which areas of the domain will need more resolution before doing the calculation.

5.2.2 Sound Diffusion Case

The next case is a sound emitted from a "speaker" in a small room covered with sound diffusion on its walls. The sound diffusion chosen is a single-plane Quadratic Residue Diffusor (QRD). QRD diffusors are composed of a series of "wells". These are square cavities of varying depths that repeat following a specific pattern that depends on the individual QRD's design. This pattern is chosen based on room size and a targeted frequency's diffusion. In this case, an N11 diffuser is be used [CD09]. This is a QRD with an 11-well pattern, shown in Figure 5.5. Each well's depth is a multiple of the depth unit, defined as $\frac{d}{11}$, where d is the design depth. To target a specific frequency for diffusion, the QRD's design depth must be half the targeted frequency's wavelength. In this case, the targeted frequency is 1000Hz, which, factoring in the speed of sound, equates to a wavelength of $\omega = 0.34$. This results in a design depth of $d = \frac{\omega}{2} = 0.17$m.

This case's domain is a rectangular room containing an array of N11 diffusors on its walls. There are 5 repetitions on the left and right boundaries, as well as 3 repetitions on the top and bottom boundaries. A small hole in the center of the domain acts as the "speaker" from which pressure waves are emitted. The initial condition is set uniformly at the flow conditions

$$\rho = 1.225 \text{ kg m}^{-3},$$

$$p = 101325.0 \text{ Pa},$$

$$u_x = 0.0 \text{ m s}^{-1},$$

$$u_y = 0.0 \text{ m s}^{-1}.$$

Boundary Conditions on the outside of the domain are set to reflection, and the boundary of the speaker is set to a varying state where pressure depends on time, mimicking the emission

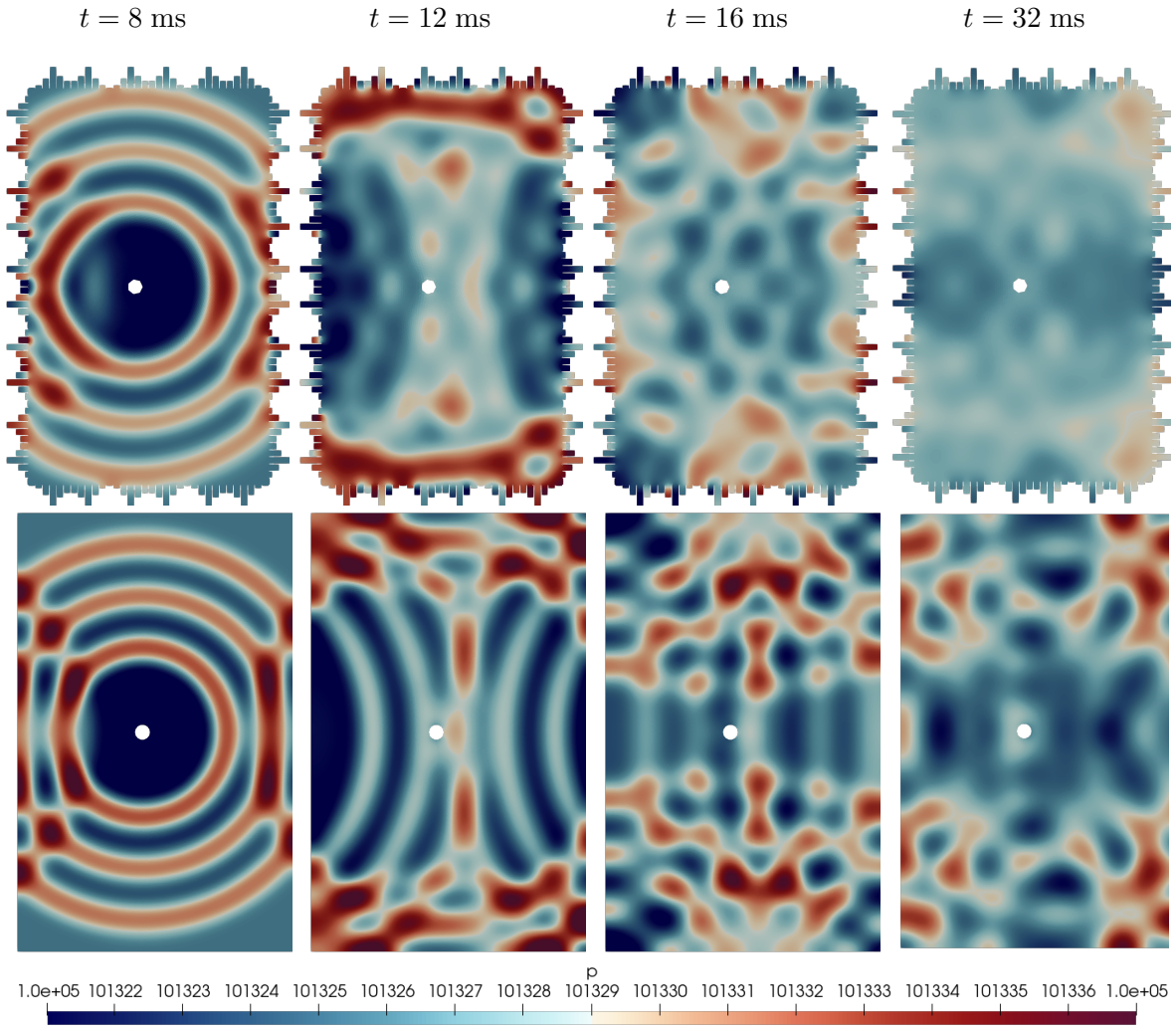

Figure 5.6: Results of the Sound Diffusion case. From left to right, the results at times 8 ms to 32 ms. The top row is the results of the room equipped with the N11 diffusor on all walls, and the bottom row shows the results on a regular rectangular room of the same size.

of sound. The pressure on the speaker's boundary is set to

$$p(t) = \begin{cases} 101325.0 - A\cos(2f\pi t) + A, & t < 0.003 \\ 101325.0, & t > 3\omega \end{cases}, \tag{5.12}$$

where amplitude $A = 40.0$ Pa, frequency $f = 1000$ Hz, and period length $\omega = 1$ ms. The initial block is refined once to create four blocks of approximately 82,000 cells each, for a total of 329,595 cells. The solution was time-stepped using a CFL number of 0.3, HLLE flux function, and Venkatakrishnan slope-limiting, to a final time of $t = 0.04$, which corresponds to 40 times the period length for a 1000z pressure wave. The final solution is shown in Figure 5.6. In Figure 5.7, the effect of the diffusor in a "listener" frame of reference is shown, if a listener were standing next to the speaker in the middle of the room (position shown with a green "X" in Figure 5.7). The red line illustrates the listener's experience in the diffusor-equipped room, and

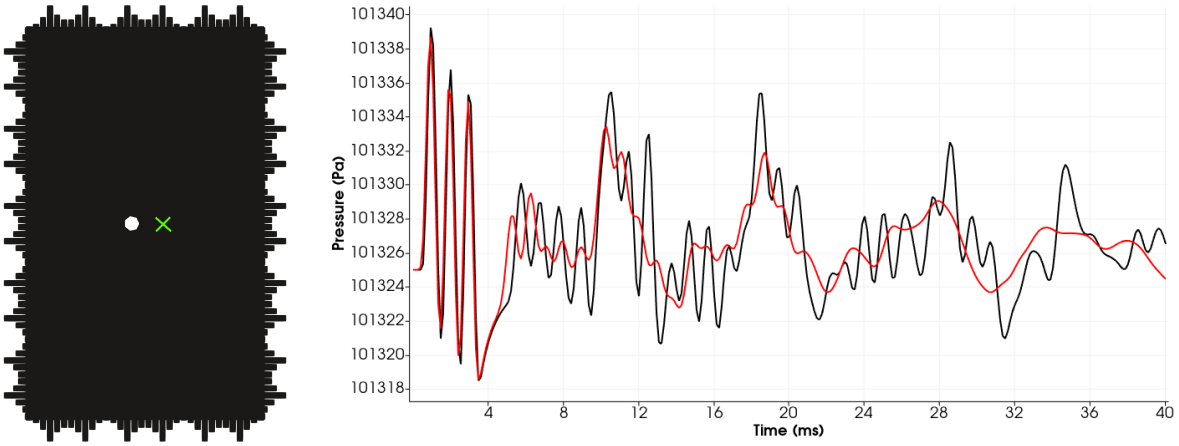

Figure 5.7: Results of the sound diffusion case at point $(1.1, 1.45)$ (pictured on the left in green) over a total of 40 ms. The black line represents the results in the regular rectangular room, and the red line represents the results in the diffusor-equipped room.

the black line pictures the listener's experience in an equivalent regular rectangular room. The pressure waves are immediately diffused after the first reflection, at approximately 4 ms. The effect is amplified with each additional reflection of the sound, with the pressure wave spikes being dampened by a factor of 3 after 3 reflections, at approximately 35 ms. The effect of the sound diffusion can also be noticed by looking at Figure 5.6. Looking at time 32 ms, the room equipped with the diffusors is coloured much more evenly than its counterpart. No significant spikes in pressure can be observed.

5.2.3 Kelvin-Helmholtz Instability

The Kelvin-Helmholtz instability occurs in any situation where an interface exists between two separate fluids that are moving relative to one another. As the fluids move relative to one another, an undulating pattern starts to appear, and grows into large wave-like structures. Vortices are created amongst these waves. This phenomenon often arises in clouds, as fast-moving lower-density air moves relative to higher-density clouds. This flow evolves to exhibit a large amount of detailed flow structure when high-accuracy numerical methods are utilized on a sufficiently resolved mesh. Hence, this case showcases the possibility to solve large problems on large computers in reasonable amounts of time using this work's code. In addition to this, this exact case was previously run on a structured mesh, and so it serves as a great comparison between the two.

The domain chosen for this situation is $x = (0, 1)$ m and $y = (0, \frac{1}{2})$ m. In this situation, the instability must be triggered by a perturbation in the interface between two fluids, which is modelled by a simple cosine function $y = 0.3 + 0.01\cos(6\pi x)$. The initial condition is pictured in Figure 5.8. The initial conditions for the top and bottom fluids are

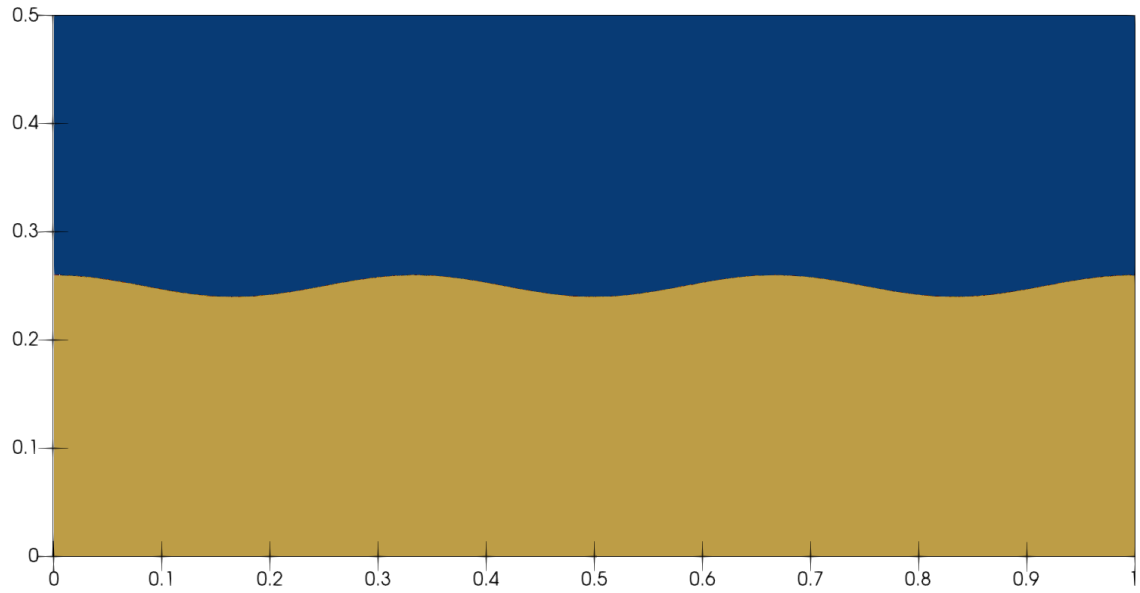

Figure 5.8: The initial condition of the Kelvin-Helmholtz instability case. "Top" fluid is in blue, and "Bottom" fluid is in yellow. The perturbation can be seen in the interface between the two fluids imparted by the cosine function.

Top	Bottom
$\rho = 1.0\ \mathrm{kg\,m^{-2}}$	$\rho = 2.0\ \mathrm{kg\,m^{-2}}$
$p = 2.5\ \mathrm{Pa}$	$p = 2.5\ \mathrm{Pa}$
$u_x = -0.5\ \mathrm{m\,s^{-1}}$	$u_x = 0.5\ \mathrm{m\,s^{-1}}$
$u_y = 0.0\ \mathrm{m\,s^{-1}}$	$u_y = 0.0\ \mathrm{m\,s^{-1}}$

(5.13)

Boundary conditions on the top and bottom of the domain are set to reflection, whereas the left and right boundaries are set to be periodic. This case is run using 768 cores, each containing a single block of approximately 51 000 cells, amounting to a total of 39 645 883 cells. The solution was time-stepped using a CFL number of 0.3, Roe flux function, and Venkatakrishnan slope-limiting, to a final time of $t = 2$ s. The final solution is shown in Figure 5.9. Despite this result being the general shape that would be expected from this type of case, if this solution is compared with the equivalent solution that was computed on a structured mesh with similar resolution, the unstructured result seems to lack detail. The structured mesh equivalent is presented in Figure 5.10, and is taken from [Kau21]. This is most likely caused by the difference in order of accuracy between structured and unstructured meshes uncovered in Section 5.1. This section describes the fact that 2nd-order accuracy is obtained on the discontinuous-Galerkin

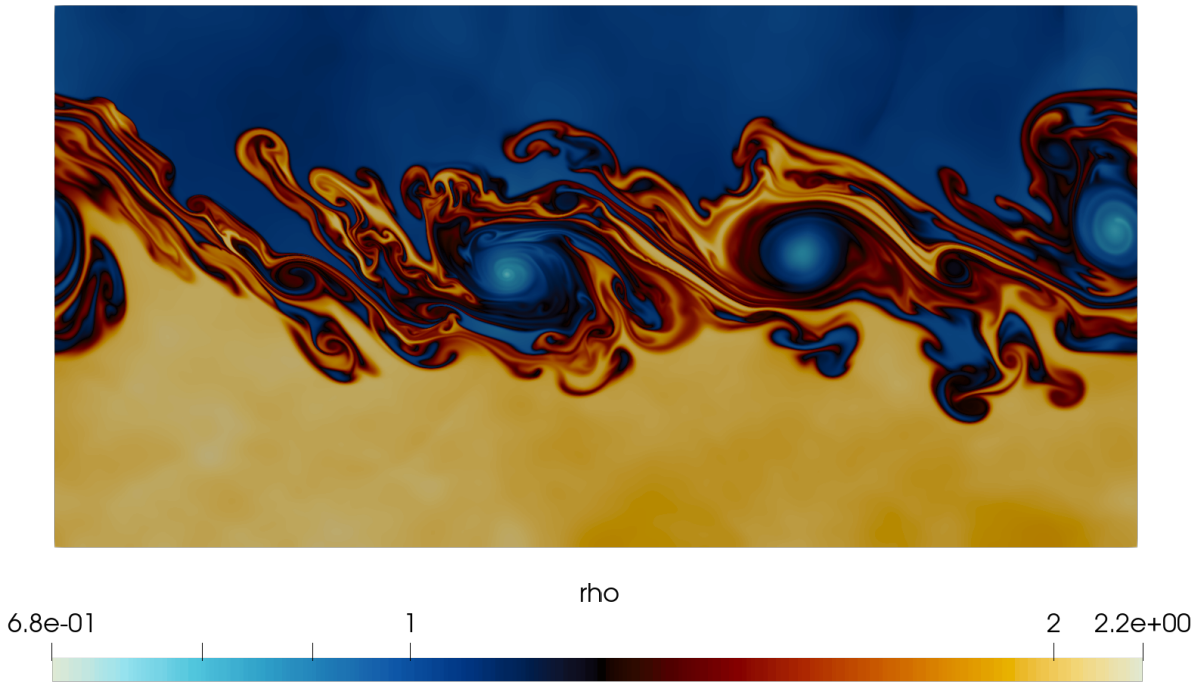

Figure 5.9: Results for the Kelvin Helmholtz Instability case, CFL number 0.3, Roe flux function, and final time of 2.0 seconds, on an unstructured mesh.

Hancock scheme on unstructured meshes, while it is 3rd-order on structured meshes.

5.2.4 3D Wing Case

This case showcases this code's ability to do 3D complex calculations. This 3D wing model was created in SolidWorks, imported into GMSH, and meshed. The airfoil profile used was the NACA 63-215, obtained from [Air]. The domain chosen is $x = (0, 3.25)$m, $y = (0, 0.9)$m, and $z = (0, 1.8)$m. The initial condition is set at

$$\rho = 1.225 \ \ \mathrm{kg\,m^{-2}}\,,$$

$$p = 101325.0 \ \ \mathrm{Pa}\,,$$

$$u_x = 102.39 \ \ \mathrm{m\,s^{-1}}\,,$$

$$u_y = 10.76 \ \ \mathrm{m\,s^{-1}}\,,$$

which is equivalent to a Mach 0.3 flow at an angle of attack of 6°. This case is run using 768 cores, and is run on a mesh containing a total of 191 828 cells. The solution is time-stepped using a CFL number of 0.3, the HLLE flux function, and Venkatakrishnan slope-limiting, to a final time of $t = 0.32$ s. The results are shown in Figure 5.11 and 5.12. In Figure 5.11 and 5.12, the streamlines can be seen being deflected in the positive-y and negative-z direction at the wing tip, which is something that could not have been noticed in a two dimensional study. This

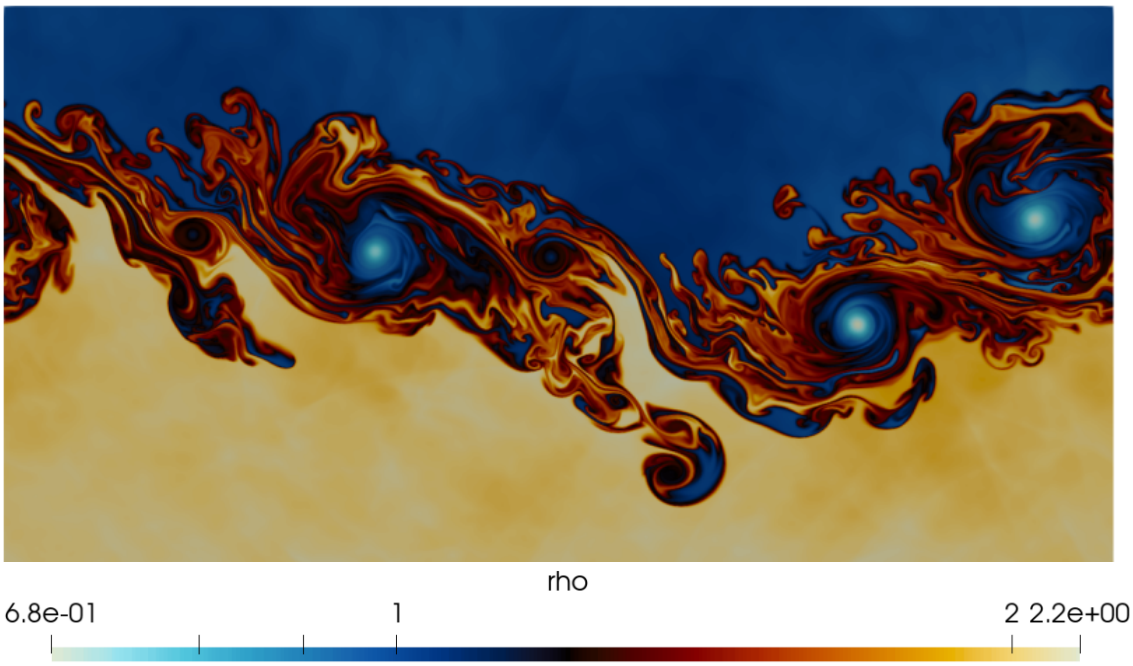

Figure 5.10: Fluid density plot for the Kelvin Helmholtz Instability case, CFL number of 0.3, Roe flux function, and final time of 2.0 seconds, on a structured mesh.

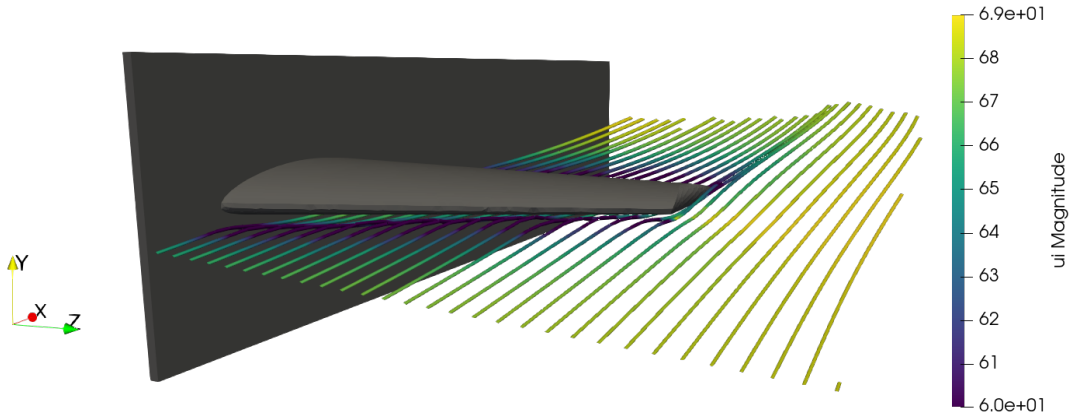

Figure 5.11: Fluid velocity plot for the 3D Wing case on airfoil profile NACA 63-215, streamlines passing over the bottom surface of the wing. Time-stepped to $t = 0.32\ s$.

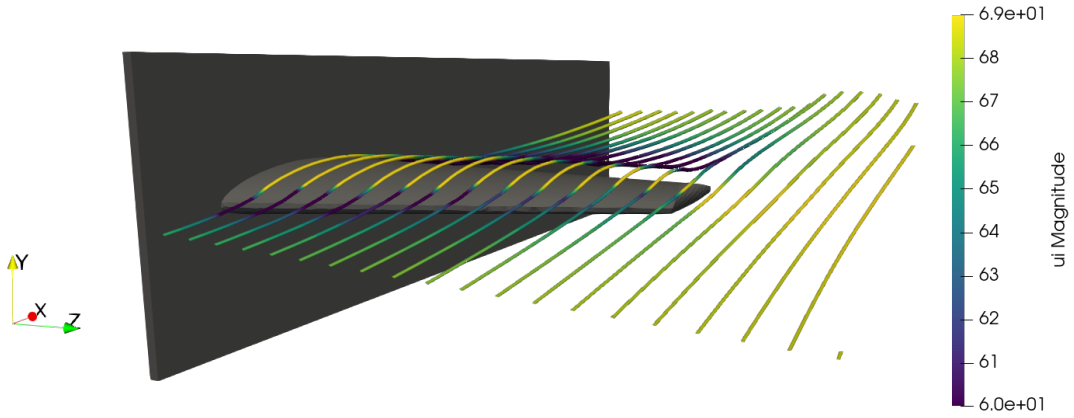

Figure 5.12: Fluid velocity plot for the 3D Wing case on airfoil profile NACA 63-215, streamlines passing over the top surface of the wing. Time-stepped to $t = 0.32\ s$.

phenomenon is what causes vortices at the wing tip, and is the sole purpose for the addition of winglets. Had this been ran using the 10-moment-model, viscosity would have amplified this effect.

5.3 The Ten Moment Model

Formulation

The 10 moment model from the kinetic theory of gases is one among many moment methods, which are described in Section 2. It assumes the velocity distribution that is used with the Boltzmann equation to be Gaussian [Lev96b]. Given that a Gaussian velocity distribution is assumed, the third-order velocity moments are zero, which translates to zero heat flux. This is a weakness of the model that will be assumed for this thesis.

To begin the 10-moment model derivation, moments of the Boltzmann equation are taken,

$$\frac{\partial}{\partial t} < WF > + \frac{\partial}{\partial x_i} < Wv_iF > + < a_iF\frac{\partial}{\partial v_i}W > = < W\frac{\delta F}{\delta t} > \ . \tag{5.14}$$

The model is obtained by assuming zero acceleration, and using the BGK collision operator [BGK54]. The moments in Equation (5.14) can be taken using the 3 weight vectors $W_1 = [m]$, $W_2 = [mv_i]$, and $W_3 = [mv_iv_j]$. This translates to the following transport equations in tensor notation,

$$\begin{aligned} m &\rightarrow \frac{\partial}{\partial t}(\rho) + \frac{\partial}{\partial x_k}(u_k) = 0 \\ mv_i &\rightarrow \frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_k}(\rho u_i u_k + P_{ik}) = 0 \\ mv_iv_j &\rightarrow \frac{\partial}{\partial t}(P_{ij}) + \frac{\partial}{\partial x_k}(u_kP_{ij}) + P_{jk}\frac{\partial}{\partial x_k}(u_i) + P_{ik}\frac{\partial}{\partial x_k}(u_j) = -\frac{1}{\tau}(P_{ij} - \frac{1}{3}P_{kk}\delta_{ij}) \end{aligned} \ . \tag{5.15}$$

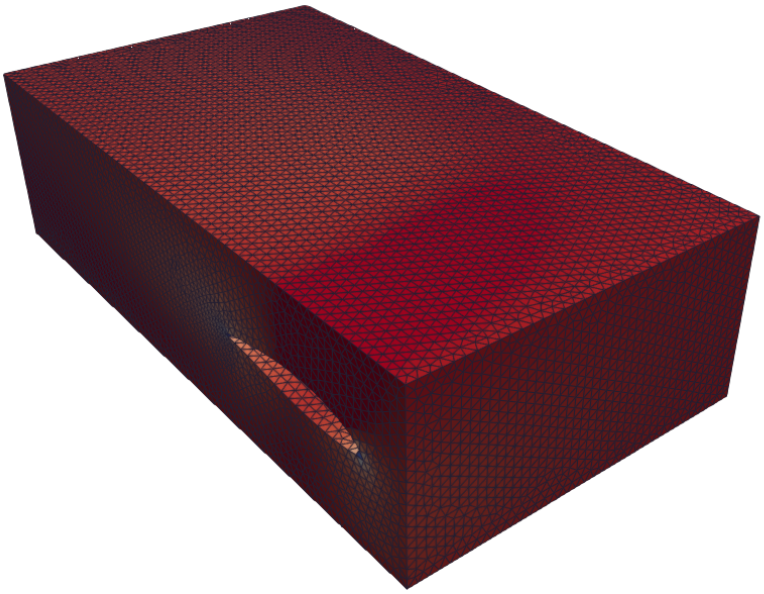

Figure 5.13: Mesh for the 3D wing case, where the full domain is shown.

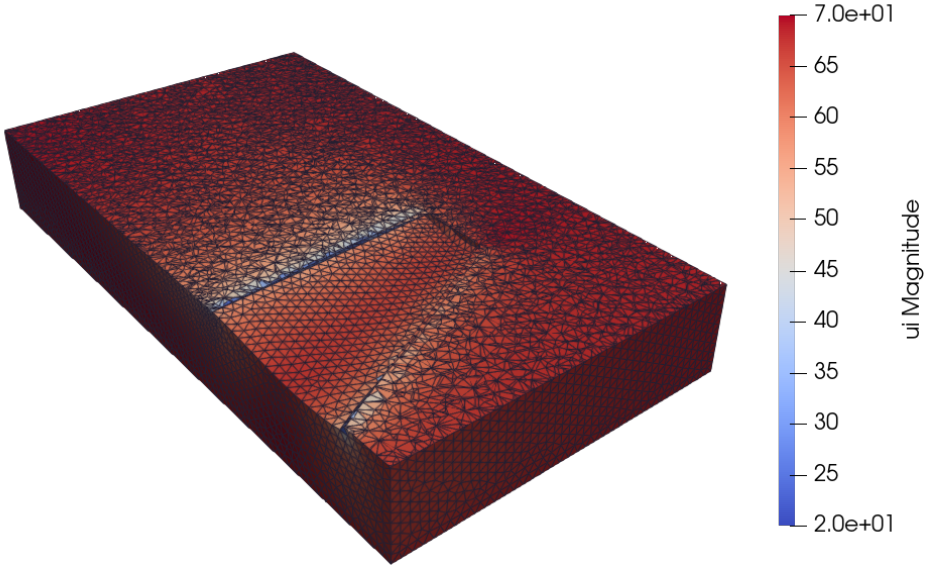

Figure 5.14: A cross section of the 3D wing case's mesh through the wing to show more of the wing's shape.

These three transport equations are each associated with a macroscopic property, ρ, u_i, and P_{ij}, respectively. The noteworthy addition in this case is P_{ij}, the pressure tensor. It allows viscosity to be captured, contrarily to the Euler equations, seen previously. In three dimensions, Equation (5.15) leads to a set of 10 transport equations, hence the name. However, this thesis only studies two-dimensional cases using the ten moment model. When in two dimensions, Equation (5.15) is a set of seven equations. For simplification, these transport equations are taken in balance law form, as

$$\frac{\partial}{\partial t}(U(x,t)) + \frac{\partial}{\partial x}(F_x(U)) + \frac{\partial}{\partial y}(F_y(U)) = S\,, \tag{5.16}$$

where U is the vector of conserved variables and S the source term, given by

$$U = \begin{bmatrix} \rho \\ \rho u_x \\ \rho u_y \\ \rho u_x^2 + P_{xx} \\ \rho u_x u_y + P_{xy} \\ \rho u_y^2 + P_{yy} \\ P_{zz}, \end{bmatrix}, \quad S = -\frac{1}{\tau} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{2P_{xx} - P_{yy} - P_{zz}}{3} \\ P_{xy} \\ \frac{2P_{yy} - P_{xx} - P_{zz}}{3} \\ \frac{2P_{zz} - P_{xx} - P_{yy}}{3} \end{bmatrix},$$

and F_x and F_y are given by

$$F_x = \begin{bmatrix} \rho u_x \\ \rho u_x^2 + P_{xx} \\ \rho u_x u_y + P_{xy} \\ \rho u_x^3 + 3u_x P_{xx} \\ \rho u_x^2 u_y + 2u_x P_{xy} + u_y P_{xx} \\ \rho u_x u_y^2 + u_x P_{yy} + 2u_y P_{xy} \\ u_x P_{zz} \end{bmatrix}, \quad F_y = \begin{bmatrix} \rho u_y \\ \rho u_x^2 + P_{xy} \\ \rho u_x u_y + P_{yy} \\ \rho u_x^2 u_y + 2u_x P_{xy} + u_y P_{xx} \\ \rho u_x u_y^2 + u_x P_{yy} + 2u_y P_{xy} \\ \rho u_y^3 + 3u_y P_{yy} \\ u_y P_{zz} \end{bmatrix}.$$

5.3.1 Multi-Element Airfoil

The first ten-moment case that was run was a multi-element airfoil, specifically the NACA 23021 with fore-flap and rear-flap, in Figure 5.16. This case was chosen to showcase the code's versatility in representing complex geometries. This case would be complex, to set up on a structured mesh given the odd geometries and multiple gaps within the mesh, where the three airfoils are located.

The domain in this case is $x = (0, 7)$ m and $y = (0, 6)$ m, with the airfoil's leading edge located at $(\frac{7}{5}, 3)$ m. The domain's initial condition is set uniformly at the Mach 0.104 flow

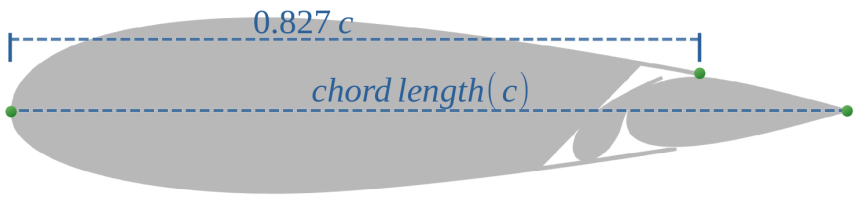

Figure 5.15: Diagram of the NACA23021 multi-element airfoil, in its retracted state. Image adapted from [FR44].

x_1	2.7%
y_1	2.45%
α_1	20°
x_2	0.2%
y_2	1.98%
α_2	40°

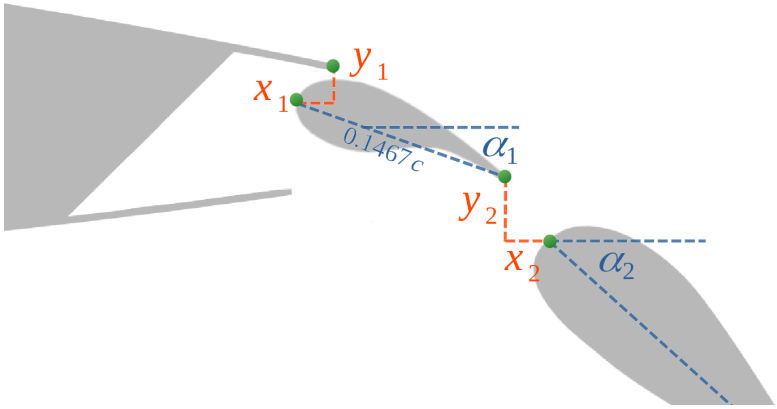

Figure 5.16: Diagram of the NACA23021 multi-element airfoil, in one of its extended states, as specified by the x, y and α values. The x and y values are expressed in percentage of the chord length, and the α values are angles relative to the chord line. Values obtained from "position 2, for $C_{d_o min}$" in [FR44].

conditions,

$$\rho = 1.225\ kg/m^2\,,$$

$$p = 101325.0\ Pa\,,$$

$$u_x = 35.6\ m/s\,,$$

$$u_y = 0.0\ m/s\,,$$

which correspond approximately to an aircraft's landing speed, a situation which would normally benefit from the added lift of a multi-element design. In this case, the Reynolds number is 2,405,405, given the 1m chord length. The solution was time-stepped using a CFL number of 0.2, HLLE flux function, and Venkatakrishnan slope-limiting, to a final time of $t = 0.03$s, enough for the values to reach steady state. The solution for an angle of attack of 4 degrees is shown in Figures 5.17 and 5.18. The lift and drag coefficients were computed using the pressure on the airfoil's boundary, and plotted for multiple angles of attack (α). The NACA 23021 multi-element design was showcased by comparing its retracted-flap and extended-flap states in Figure 5.19. There are a few things to note about these airfoil results. Although the ten moment model is a good option for many types of flows, this specific case has a significant dependence on turbulence modelling. No turbulence model has been developed to complement the ten moment model for these types of situations as of yet. This means the boundary layer and flow separation are not predicted with significant accuracy. A significant advantage of the

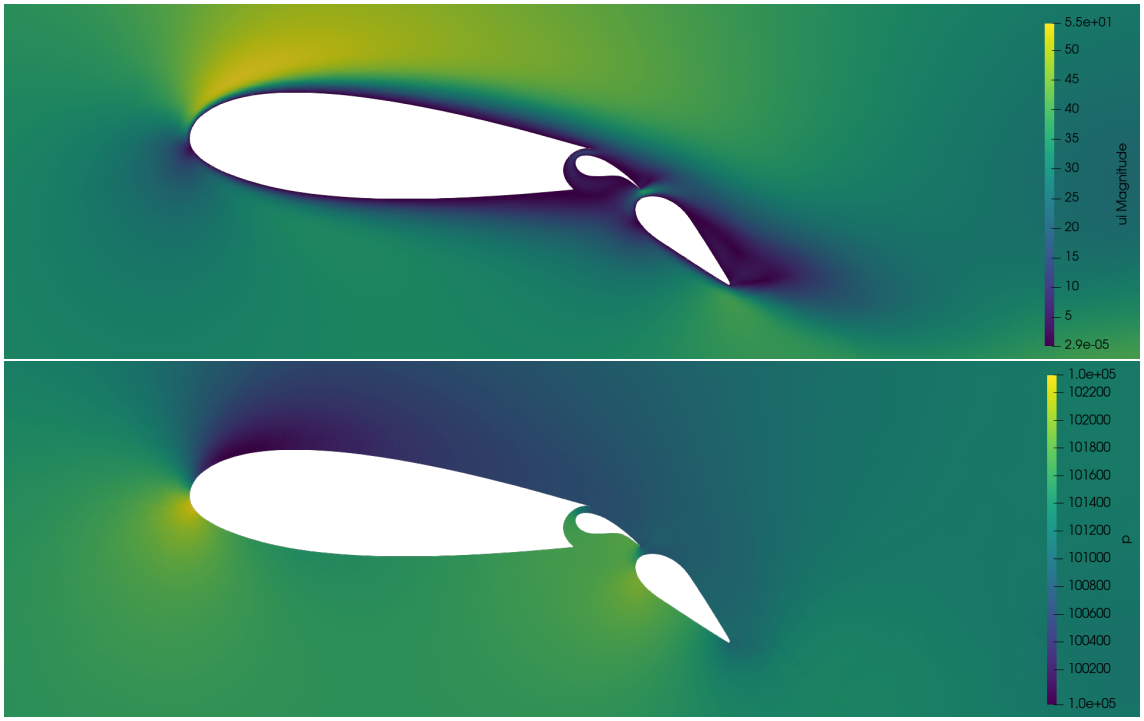

Figure 5.17: The pressure and flow speed results for the NACA23021 multi-element airfoil in its extended state, at angle of attack $\alpha = 4°$.

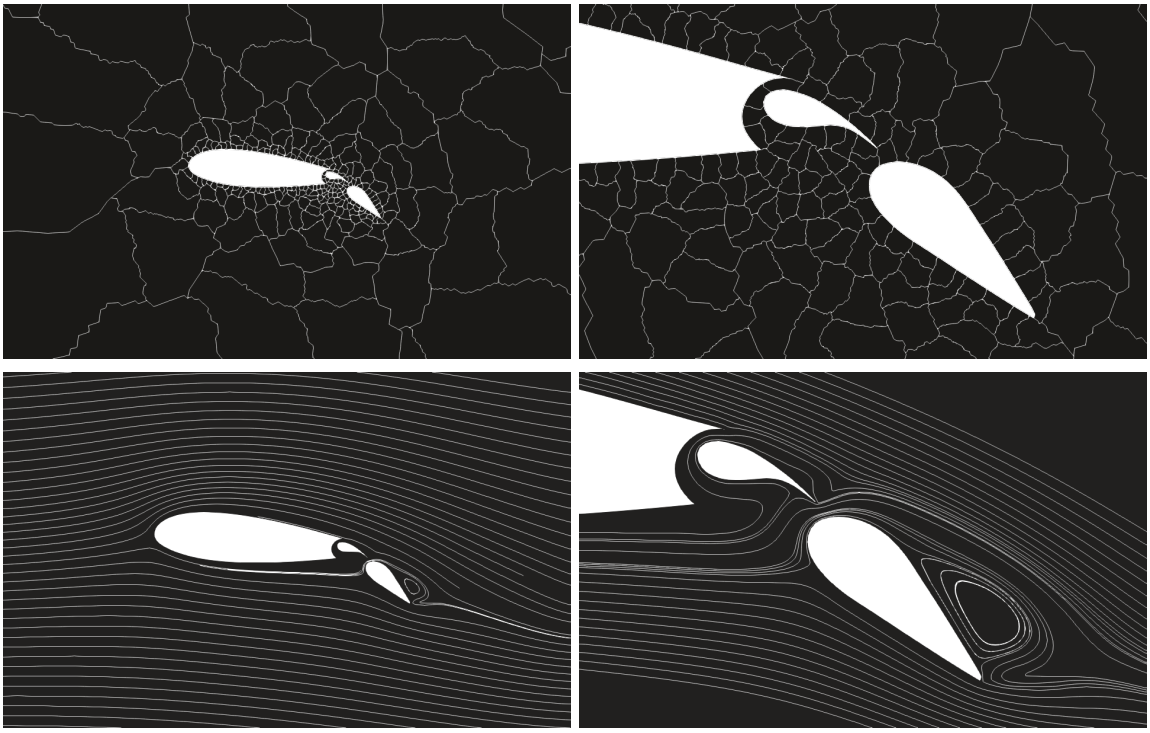

Figure 5.18: Results for the NACA 23021 Multi-Element Airfoil at angle of attack $\alpha = 4°$. The top images show the block boundaries, each of the blocks containing 5500 cells. The bottom two images show the streamlines around the airfoil.

	Extended		**Retracted**	
α(deg)	C_d	C_l	C_d	C_l
-8	0.153	0.649	-	-
-4	0.190	0.997	-	-
0	0.252	1.33	0.075	0.092
4	0.337	1.661	0.093	0.359
8	0.452	1.966	0.131	0.630
16	-	-	0.265	1.14

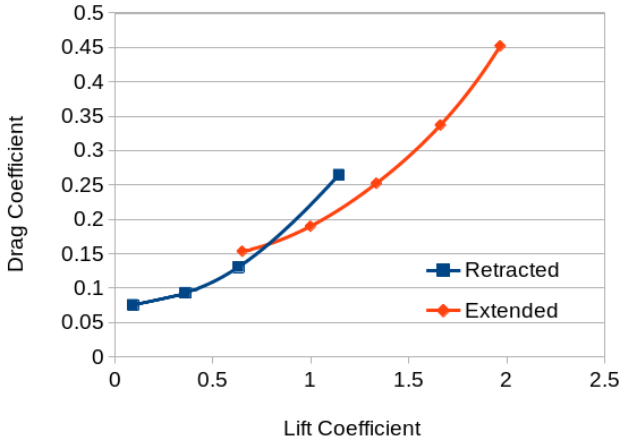

Figure 5.19: C_l and C_d results for angles of attack α between -8 and 16, for both the Extended and Retracted states of the NACA23032 multi-element airfoil.

multi-element design is its ability to use high-energy flow from the bottom side of the wing to re-energize the boundary layer on the top side of the wing. Despite these inaccuracies, relatively good results are obtained for the lift and drag coefficients when comparing the extended position to the retracted position. The relation between the two resembles experimental results [FR44] in shape.

Chapter 6

Conclusions

6.1 Summary

The current work presents the implementation of unstructured meshes in a scheme for solving general hyperbolic-relaxation partial differential equations in and out of thermodynamic equilibrium. The highly parallelizable discontinuous-Galerkin Hancock scheme is presented, starting with its formulation. A subsection describes the integration methods used in the scheme, and another describes the intercell flux calculation methods.

The 2D and 3D implementation of unstructured partitions is also presented. Output from gmsh is read in, a meshing software which is used to create our 2D and 3D geometry and generate meshes. METIS, a software package used for graph partitioning, is used in the 2D and 3D refinement process along with cell sub-splitting to achieve a finer mesh. This mesh refinement is triggered adaptively during the running computation, depending on chosen parameters within a part of the mesh, such as the l^2 norm of the gradient. Gaussian quadrature is used to compute cell geometrical properties and compute integrals within cells and boundaries.

A number of results obtained using the DGH scheme with unstructured partitions and different PDEs both in 2D and 3D are presented. Many of these incorporate refinement or adaptive refinement, which allows computations to occur efficiently on modern distributed clusters. Results obtained include supersonic flow over a bump, flow over a multi-element airfoil, sound waves in a diffusion-equipped room, and more.

Using the linear convection equations, the scheme was proven to be 2nd-order accurate for 2D and 3D, despite the 3rd-order expectation. Although a bug in the code can never be ruled out, extensive verification was done, including quadrature rule verification, quadrature point verification, and geometrical property verification.

The implementation was shown to scale efficiently on large clusters using a strong scaling analysis with a computationally expensive Kelvin-Helmholtz instability case.

6.2 Future Work

The current work definitely allows room for improvement. For the scheme and PDEs, the following improvements could be made:

- Revisiting the DGH scheme to verify the reason for the lack of 3rd-order accuracy in unstructured meshes.
- Implementing a turbulence model in the 10 moment method to allow accurate results where turbulence is prominent. This work is currently ongoing, as seen in recent work by Yan [YM22].

As for the unstructured partition implementation, the following improvements could be made:

- Given the fact that the solution is computed simply by iterating over all mesh boundaries' quadrature points to compute cell updates, it would be relatively straightforward to allow for cells that have more faces than triangles or tetrahedra. This would allow more flexibility in complex geometries.
- Although it would be quite challenging on an unstructured partition, implementing adaptive coarsening alongside the adaptive refinement would provide ultimate flexibility when it comes to live alterations to the mesh's resolution.
- Now that both structured and unstructured meshes are implemented within this code, hybrid structured-unstructured meshes could be a possibility in the near future, as there are many advantages to using unstructured meshes for parts of the mesh that have unusual geometry, and using structured meshes for the other parts, speeding up the calculations.

References

[FR44] J. Fischel and J. M. Riebe. *Wind-Tunnel Investigation of an NACA 23021 Airfoil with a 0.32-Airfoil-Chord Double Slotted Flap*. Tech. rep. National Advisory Committee for Aeronautics, Oct. 1944.

[Gra49] Harold Grad. "On the kinetic theory of rarefied gases". In: *Communications on pure and applied mathematics* 2.4 (1949), pp. 331–407. ISSN: 0010-3640.

[BGK54] P. L. Bhatnagar, E. P. Gross, and M. Krook. "A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems". In: *Physical Review* 94.3 (1954), pp. 511–525. ISSN: 0031-899X.

[RH73] W H Reed and T R Hill. "Triangular mesh methods for the neutron transport equation". In: (Oct. 1973). URL: https://www.osti.gov/biblio/4491151.

[LR74] P. LESAINT and P.A. RAVIART. "On a Finite Element Method for Solving the Neutron Transport Equation". In: *Mathematical Aspects of Finite Elements in Partial Differential Equations*. Ed. by Carl de Boor. Academic Press, 1974, pp. 89–123. ISBN: 978-0-12-208350-1. DOI: https://doi.org/10.1016/B978-0-12-208350-1.50008-X.

[Roe81] P. L. Roe. "Approximate Riemann solvers, parameter vectors, and difference schemes". In: *Journal of Computational Physics* 43 (1981), pp. 357–372.

[HLL83] A. Harten, P. D. Lax, and B. Van Leer. "On Upstream Differencing and Godunov-Type Schemes for Hyperbolic Conservation Laws". In: *SIAM Review* 25.1 (1983), pp. 35–61.

[JP86] C. Johnson and J. Pitkäranta. "An Analysis of the Discontinuous Galerkin Method for a Scalar Hyperbolic Equation". In: *Mathematics of Computation* 46.173 (1986), pp. 1–26. ISSN: 00255718, 10886842.

[Ein88] Bernd Einfeldt. "On Godunov-Type Methods for Gas Dynamics". In: *SIAM Journal on Numerical Analysis* 25.2 (1988), pp. 294–318. DOI: 10.1137/0725021.

[Sch91] W. E. Schiesser. "The Numerical Method of Lines: Integration of Partial Differential Equations". In: (1991).

[Gom94] Tamas I. Gombosi. *Gaskinetic Theory*. Cambridge Atmospheric and Space Science Series. Cambridge University Press, 1994. DOI: 10.1017/CBO9780511524943.

[Ven95] V. Venkatakrishnan. "Convergence to Steady State Solutions of the Euler Equations on Unstructured Grids with Limiters". In: *Journal of Computational Physics* 118.1 (1995), pp. 120–130. ISSN: 0021-9991.

[Lev96a] C. Levermore. "Moment closure hierarchies for kinetic theories". In: *Journal of Statistical Physics* 83 (June 1996), pp. 1021–1065. DOI: 10.1007/BF02179552.

[Lev96b] David C. Levermore. "Moment closure hierarchies for kinetic theories". In: *Journal of Statistical Physics* 83.5-6 (1996), pp. 1021–1065. ISSN: 0022-4715.

[CS98] Bernardo Cockburn and Chi-Wang Shu. "The Runge–Kutta Discontinuous Galerkin Method for Conservation Laws V: Multidimensional Systems". In: *Journal of Computational Physics* 141.2 (1998), pp. 199–224. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1998.5892.

[KK99] G. Karypis and V. Kumar. "A fast and high quality multilevel scheme for partitioning irregular graphs". In: *SIAM Journal on Scientific Computing* 20.1 (1999), p. 359.

[Suz08] Y. Suzuki. "Discontinuous Galerkin Methods for Extended Hydrodynamics". PhD thesis. The University of Michigan, Aerospace Engineering and Scientific Computing, 2008.

[CD09] T.J. Cox and P. D'Antonio. *Acoustic Absorbers and Diffusers: Theory, Design and Application*. Taylor & Francis, 2009. ISBN: 9780203893050. URL: https://books.google.ca/books?id=f19_6NFg4jkC.

[GR09] C. Geuzaine and J.-F. Remacle. "Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities". In: *International Journal for Numerical Methods in Engineering* 79.11 (2009), pp. 1309–1331.

[Kau21] W. Kaufmann. "Extended Hydrodynamics using the Discontinuous-Galerkin Hancock Method". The University of Ottawa, Department of Mechanical Engineering, 2021.

[YM22] Chao Yan and James G. McDonald. "First-order hyperbolic-relaxation turbulence modelling for moment-closures". 2022.

[Air] AirfoilTools. *NACA 63-215 AIRFOIL (n63215-il)*. Accessed: 2023-05-22. URL: http://airfoiltools.com/airfoil/details?airfoil=n63215-il.

[Bura] J. Burkardt. *Computational Geometry Lab: MAPPING TETRAHEDRONS*. Computational Geometry Lab. Accessed: 2023-02-20.

[Burb] J. Burkardt. *QUADRATURE_RULES_TRI*. Accessed: 2023-04-15. URL: http://people.sc.fsu.edu/~jburkardt/datasets/quadrature_rules_tri/quadrature_rules_tri.html.