NUMERICAL SIMULATION
OF MOVING BOUNDARY PROBLEMS
RELATED TO FRACTURE

Graeme H. Morgan

CORPUS CHRISTI COLLEGE
UNIVERSITY OF CAMBRIDGE

A DISSERTATION
SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE
MASTER OF PHILOSOPHY

August 2013
CONTENTS

Contents i
Summary iii
Preface v
1 Introduction 1
2 Riemann solvers 5
  2.1 The inert Euler equations 5
  2.2 Hyperbolicity 6
  2.3 The Riemann problem 7
    2.3.1 The HLLC approximate Riemann solver 7
  2.4 Numerical method 8
    2.4.1 Operator splitting 9
    2.4.2 MUSCL–Hancock scheme 10
    2.4.3 Higher-order non-MUSCL methods 12
    2.4.4 CFL condition 13
3 CFD on the GPU 15
  3.1 Processing architecture 16
  3.2 Suitable problems 16
  3.3 Compute Unified Device Architecture (CUDA) 17
4 Ghost fluid method 19
  4.1 Signed distance function 19
  4.2 Results 21
    4.2.1 Static cylinder 21
    4.2.2 Steady-state aerofoil flow 25
    4.2.3 Mass loss 27
## Cut cell methods

5.1 Notation .................................................. 31
5.2 Description of geometry .............................. 32
5.3 The $h$-box method ................................. 33
   5.3.1 Non-orthogonal rotated grid method .......... 34
   5.3.2 Construction of $h$-boxes ...................... 35
   5.3.3 Slope reconstruction and limiting ............ 37
   5.3.4 Problems with the $h$-box method .......... 39
5.4 Results ............................................. 40
   5.4.1 Static cylinder .................................... 40
   5.4.2 Steady-state aerofoil flow .................... 43
   5.4.3 Sloped channel flow ............................ 43

## Moving cut cell methods

6.1 General method ..................................... 45
6.2 Piston flux applied naïvely to the $h$-box method . 47
6.3 Cell merging & the rotated grid method .......... 49
6.4 Results ............................................. 50

## Conclusions

7.1 Stationary cut cell methods ....................... 53
7.2 Moving cut cell methods ........................... 54
7.3 Further work ...................................... 54

## References

References .............................................. 55
This thesis presents a review of numerical methods for modelling flow around immersed solid boundaries. The majority of such schemes offer only first-order convergence in the vicinity of boundaries; this is not always a significant problem, as the lower dimensionality of the interface often enables higher order convergence in the domain as a whole. However, for simulations involving crack propagation and fracture, in which a significant interface length may be introduced, we desire higher order convergence and explicit conservation.

The $h$-box method of Berger and Helzel (2012) is found to be the only method offering second-order convergence at boundaries. The method is implemented, the convergence properties verified, and applied to a number of test problems with static interfaces. We show that the method has substantial advantages over the ghost fluid method in reducing waves generated by the boundary approximation.

The remainder of the work combines the principles of the $h$-box method with a cell merging algorithm to construct a cut cell method for moving immersed boundaries. This new method shows promise of super-linear convergence up to a certain grid resolution.
I am indebted to my supervisor, Dr. Nikolaos Nikiforakis, for his patient guidance through the murky waters of Computational Fluid Dynamics; to Dr. Alan Minchinton for his support, especially when changing the main focus of this dissertation; and to Orica for funding this research. None of this would have been possible without the support of my partner, Alison Ming, who has kept me sane while slicing and dicing cells.

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text. None of this work has been previously submitted in part or in whole for any other degree at this or any other university. In addition, this dissertation is approximately 10,000 words in length.

Changes made since submission:

• Minor error in Equation 4.13 fixed (24/09/2013)
• Missing figure restored (Figure 5.7) (24/09/2013)
INTRODUCTION

Numerical simulation of rock fracture and fragmentation is a topic of great interest to the mining industry. The experiences of individual miners are no longer sufficient to safely and accurately handle complex blast problems; instead, quantitative computer models are required. Recent years have seen significant investment in computational models with the primary goals of avoiding damage to structures near the blast site, and predicting the distribution of resources in the blast pile.

A large number of individual processes need to be captured for a complete simulation: ideally the detonation of explosives should be modelled by a reactive code, which must then be coupled to an elasto-plastic model of the rock to capture deformations. The latter must include the ability to model fracture, damage, and solid body heave of rock. Depending on the problem, it may also be necessary to handle the in-flight interactions of ejected rock fragments.

The majority of current models are based on finite difference or finite element methods. However, shock-capturing detonation codes typically use finite volume methods, leading to problems coupling the two. Recent work extends the commercially available ELFEN code with Orica’s CPeX non-ideal detonation code which provides the borehole pressure history to the former. However, a fully-coupled model which captures the full interaction between the detonation front and the surrounding rock is desired; strides toward this goal are described in Schoch et al. (2013) in which a non-ideal mining explosive is modelled with an elasto–plastic confiner in a finite volume scheme.

However, modelling fracture in finite volume codes provides a new set of problems.
Before attempting to integrate a fracture model, we require a system for dealing with
the boundaries between different phases which will be introduced by cracks. Current
codes within the Laboratory for Scientific Computing use the ghost fluid method to
solve for multimaterial flows, even with sharp density gradients (such as air–water
interfaces). This approach is widely used because of significant advantages, including
robustness and ease of implementation.

A main disadvantage is its behaviour at boundaries, where it is both non-conservative
and reduces to first-order even when using a second-order numerical method. For some
problems the poor convergence properties are not a problem, as the boundary is of
lower dimensionality to the main body of the fluid; this means that the ratio of bound-
ary cells to normal cells typically scales as \(1/N\). However, if a significant number of
fragments are produced, this assumption breaks down, and it is likely to cause an over-
all reduction to first-order. A ghost fluid method slight improvement over the original
method is described in Chapter 4, and applied to a variety of test problems, including
a quantification of conservation error.

![Figure 1.1: A sample simulation which demonstrates the issue of mass loss. The initial
geometry consists of 20 pre-split fragments of solid separated by small gaps, each moving
outward radially with different velocities. By the time the fragments reach the edge of
the domain (which is \(10\times\) the fragment length, and discretised into \(250 \times 250\) cells), the
mass error is 1%.](chart)

Cut cell methods provide a conservative and typically more accurate alternative for
representing immersed material boundaries. The zeroth-order\(^1\) representation of the
boundary is replaced with a piecewise-linear description. This produces irregular cells
in the vicinity of the boundary; the problem to be overcome is that these cells can be
arbitrarily small, which will reduce the stable time step significantly. Even if we could
accept a time step orders of magnitude smaller than that determined by the regular

---

\(^1\)That is, step-wise, with each cell flagged as inside or outside the solid boundary.
grid spacing, the resulting numerical diffusion would become ruinous.

A description of cut cell methods is provided in Chapter 5, and the $h$-box method from Berger and Helzel (2012) is chosen as a starting point for further research. We show that this fully second-order cut cell method allows us to overcome the problems of the ghost fluid method,

1. spurious waves arising from the interaction of a fluid with the stepped boundary,
2. lack of conservation due to flux across boundary cells.

Problems involving boundaries with motion relative to the computational grid are easily handled using the ghost fluid method, but more limited work has been presented using cut cells. Crucially, Barton et al. (2011) describes a multimaterial interface-tracking method which, like the ghost fluid method, uses a level set to track the boundary between phases, but then reconstructs a cut cell mesh and implements a conservative solution. Chapter 6 describes progress towards a new moving boundary method based on the rotated grid method (Helzel et al., 2005) and a standard cell-merging scheme. The overarching aim of this work is a future implementation of a superlinear and inherently conservative cut cell algorithm which can be used in place of the ghost fluid method.

As part of this work, we,

• provide a brief review of the main literature on ghost fluid and cut cell methods,
• produce an algorithm and implement a GPU code for both static and moving solid body problems using the ghost fluid method,
• produce a GPU implementation of the $h$-box cut cell method, and validate that it provides second-order convergence at boundaries,
• test the $h$-box method for static boundary methods which cause the ghost fluid method to produce spurious oscillations,
• based on the principles of the $h$-box method, implement a new moving cut cell method which is explicitly conservative and super-linear up to a point.
Throughout this work, gas flow will be modelled by the Euler equations. We require a compressible model to capture shock waves, but due to the high velocity of the flow in most problems, viscosity can be neglected\(^1\).

### 2.1 The Inert Euler Equations

In conservative form, the compressible Euler equations are,

\[
\begin{align*}
\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) &= 0, \\
\frac{\partial \rho u}{\partial t} + \nabla \cdot (\rho uu^T) + \nabla p &= 0, \\
\frac{\partial E}{\partial t} + \nabla \cdot (u(E + p)) &= 0,
\end{align*}
\]

(2.1a, 2.1b, 2.1c)

where \(\rho\) is the gas density, \(u\) is velocity, \(p\) is pressure, and \(E\) is the total energy density related to the internal energy density \(e\) by,

\[
E = \rho e + \frac{1}{2}\rho |u|^2.
\]

(2.2)

However, this gives (in one dimension) three equations and four quantities for which we must solve. As such, the system must be closed by one further equation. For this

\(^1\)In principle, it should be possible to restore viscosity using a hyperbolised Navier–Stokes approach.
work, as the focus is on the numerical scheme, the ideal gas equation of state will be used for simplicity, given by,
\[ p = \rho e (\gamma - 1), \quad (2.3) \]
in “caloric form”, where \( \gamma = \frac{c_p}{c_v} \) is the adiabatic index. In terms of the number of degrees of freedom, \( n, \gamma = \frac{n + 2}{n} \); for a diatomic gas at room temperature \( \gamma = 1.4 \), dropping to \( \gamma = 1.33 \) at high temperatures when additional rotational degrees of freedom are accessible.

### 2.2 Hyperbolicity

We may rewrite the one dimensional case of Equation 2.1 in the more general conservation law form,
\[ \frac{\partial U}{\partial t} + F(U)_x = 0, \quad (2.4) \]
which can be recast in terms of the Jacobian \( J(U) = \frac{\partial F}{\partial U} \) to give,
\[ \frac{\partial U}{\partial t} + A(U)U = 0. \quad (2.5) \]

**Definition:** The \( n \times n \) system of conservation laws given by Equation 2.5 is said to be hyperbolic if \( A(U) \) has \( n \) real eigenvalues and a corresponding set of \( n \) linearly independent eigenvectors \( K^{(i)} \).

The Euler equations are provably hyperbolic with equations of state which describe the ideal gas, as well as more complex materials (Toro, 1999). Hyperbolicity implies that the matrix \( A \) can be diagonalised in its characteristic basis,
\[ A = K \Lambda K^{-1}, \quad (2.6) \]
where \( \Lambda = \text{diag}(\lambda_1, ..., \lambda_n) \) is a diagonal matrix of eigenvalues, and \( K \) is a matrix with columns \( K^{(i)} \). We can then rewrite Equation 2.4 in terms of its characteristic variables \( W = K^{-1}U \) to obtain a system of \( n \) decoupled equations,
\[ \frac{\partial W_i}{\partial t} + \lambda_i \frac{\partial W_i}{\partial x} = 0. \quad (2.7) \]

The differential equation for each member of \( W \) can be identified as a linear advection equation. It follows that the solution to a set of hyperbolic conservation laws consists of \( n \) waves moving with speeds given by the eigenvalues \( \lambda_1, ..., \lambda_n \). For the Euler equations, these speeds are \( u - c, u, u + c \), where \( u \) is the fluid speed and \( c \) the speed of sound. We will see later that this property of locality created by the finite speed of propagation of waves is of utmost importance when parallelising hyperbolic equation codes.
2.3 THE RIEMANN PROBLEM

The Riemann problem for a set of hyperbolic conservation laws deals with the solution for \( t > 0 \) with initial conditions which are discontinuous at the origin,

\[
U(t = 0, x) = \begin{cases} 
U_L & x \leq 0, \\
U_R & x > 0. 
\end{cases}
\] (2.8)

A vast amount of information about the problem is gained from analytically solving the Riemann problem, but would be repetitious; a full description can be found in Toro (1999). However, by regarding the numerical solution \( U_i^n \) to be piecewise constant across each cell, as illustrated in Figure 2.1, the Riemann problem can be solved at time \( t = \Delta t \) with initial conditions \( U_i^n \) and \( U_{i+1}^n \) to obtain the solution at \( U_{i+\frac{1}{2}}^n \), which can then be used to compute the inter-cell flux \( F_{i+\frac{1}{2}} \). Using Equation 2.18 combined with the fluxes at each cell boundary allows for the computation of \( U_i^{n+1} \).

The structure of the Riemann problem consists of left- and right-going shock or rarefaction waves in various allowed combinations, separated by a contact surface.

2.3.1 THE HLLC APPROXIMATE RIEMANN SOLVER

The Harten–Lax–van Leer (HLL) (Harten et al., 1983) approximate Riemann solver forms estimates for the speeds of the left- and right-going waves based on the sonic
Numerical method

velocity and gas velocity,

\[ S_L = \min \left( (u - c)_{\text{Roe}}, u_L - c_L \right), \]
\[ S_R = \min \left( (u + c)_{\text{Roe}}, u_R + c_R \right), \]

where the subscript Roe denotes the averaging,

\[ \langle \cdot \rangle_{\text{Roe}} = \frac{\sqrt{\rho_L}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \langle \cdot \rangle_L + \frac{\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \langle \cdot \rangle_R. \]

An estimate for the speed of the contact surface was restored in Toro et al. (1994) to form the HLLC solver,

\[ S^* = p_R - p_L + \rho_L u_L (S_L - u_L) - \rho_R u_R (S_R - u_R) \]
\[ \rho_L (S_L - u_L) - \rho_R (S_R - u_R) \]

which together with an estimated “star state” between the left and rightmost waves,

\[ U^*_k = \frac{S_k - u_k}{S_k - S} \left( \begin{array}{c} \rho_k \\ \rho_k S^* \\ \frac{\rho_k S^* + \frac{p_k}{S_k - u_k}}{E_k + (S_k - u_k)} \end{array} \right), \]

allows for the simple expression of the HLLC flux between two cells, where each

\[ F_{\text{HLLC}}(U_L, U_R) = \begin{cases} F(U_L), & 0 \leq S_L, \\ F(U_L) + S_L (U_{sL} - U_L), & S_L \leq 0 \leq S^*, \\ F(U_R) + S_R (U_{sR} - U_R), & S^* \leq 0 \leq S_R, \\ F(U_R), & S_R \leq 0. \end{cases} \]

While exact solutions of the one dimensional Riemann solver are not difficult to compute, doing so is around 20× slower than using an approximate solver, and the differences in result are generally negligible.

2.4 Numerical method

In order to numerically solve the Euler equations (Equation 2.1), we employ a finite volume formulation in which the value at each point on the discrete grid \( U^n_i \) is interpreted as the average over a single cell,

\[ U^n_{i,...,k} = \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{x_{k-1/2}}^{x_{k+1/2}} U(n \Delta t, x, \ldots, z) \, dx \ldots dz 
= \int_V U(n \Delta t, x) \, dV. \]
Riemann solvers

Taking the system of equations in conservative form,

$$\frac{\partial U}{\partial t} = -\nabla \cdot F(U), \quad (2.16)$$

and integrating over a single cell volume \( V \), with application of the Leibniz integral rule (as \( V \) is constant), and Stokes’ theorem, an expression for the rate of change of the cell average is recovered,

$$\frac{d}{dt} \int_V U(n \Delta t, x) \, dx = - \int_V \nabla \cdot F(U) \, dV,$$

$$\frac{d}{dt} U^n = - \int_{\partial V} F(U) \cdot dr. \quad (2.17)$$

As the integral over the surface of the cell \( \partial V \) consists only of two points in one dimension, at the front and back of the cell, further discretising the time integral leads to,

$$U^{n+1}_i = U^n_i - \Delta t \Delta x \left[ F\left(U^n_{i+\frac{1}{2}}\right) - F\left(U^n_{i-\frac{1}{2}}\right) \right]. \quad (2.18)$$

### 2.4.1 Operator splitting

In two dimensions, the system of equations can be written as,

$$\frac{\partial U}{\partial t} + L_x U + L_y U = 0, \quad (2.19)$$

with,

$$L_x U = \frac{\partial}{\partial x} \begin{pmatrix} \rho u \\ \rho u^2 \\ \rho u v \\ u (E + p) \end{pmatrix}, \quad L_y U = \frac{\partial}{\partial y} \begin{pmatrix} \rho v \\ \rho u v \\ \rho v^2 \\ v (E + p) \end{pmatrix}, \quad (2.20)$$

where \( u = (u, v) \). Rather than solving the entire set of operators at once for each time step, which would be mathematically complex, each one is advanced individually while ignoring the others. First, \( L_x \) is solved with a one dimensional Riemann solver, followed by \( L_y \),

$$\frac{\partial U}{\partial t} + L_x U = 0, \quad U^n \rightarrow U^{n+\frac{1}{2}},$$

$$\frac{\partial U}{\partial t} + L_y U = 0, \quad \underbrace{U^{n+\frac{1}{2}}}_{\text{ICs}} \rightarrow \underbrace{U^{n+1}}_{\text{output}} \quad (2.21)$$

where the fractional time indices denote the progression within one time step rather than shorter time steps.
2.4.2 MUSCL–Hancock scheme

Unfortunately, using Godunov’s method with piecewise constant data is only first-order accurate in space. There are two main approaches to improving the rate of convergence:

1. we can use more information from the Riemann solution, as in the Weighted Average Flux method, which allows for second-order accuracy,
2. the more usual method, which in principle allows for better than second-order accuracy, involves constructing values at the cell boundaries which are better than piecewise constant.

The MUSCL–Hancock scheme reconstructs piecewise linear data across each cell. Defining the forward difference between adjacent cells as,

$$\Delta U_{i+\frac{1}{2}} = U_{i+1} - U_{i},$$

$$\Delta U_{i-\frac{1}{2}} = U_{i} - U_{i-1}, \quad (2.22)$$

and the centred difference across each cell as,

$$\Delta_i = \Delta U_{i+\frac{1}{2}} + \Delta U_{i-\frac{1}{2}}, \quad (2.23)$$
then the state vector is extrapolated to the left and right boundaries of each cell using,

\[ \mathbf{U}_i^L = \mathbf{U}_i - \frac{1}{2} \mathbf{\Delta}_i, \]
\[ \mathbf{U}_i^R = \mathbf{U}_i + \frac{1}{2} \mathbf{\Delta}_i, \]  
(2.24)

where \( \mathbf{\Delta}_i \) is a limited slope. By Godunov’s theorem (Godunov, 1959), direct application of a second-order scheme leads to spurious oscillations around discontinuities. This problem is avoided by applying a first-order method where the solution changes rapidly, and the full second-order method in smooth regions. The limited slope is defined as,

\[ \mathbf{\Delta}_i = \phi(r) \Delta_i \]  
(2.25)

where vector notation is somewhat abused; the function \( \phi(r) \) should be taken to apply to each element of \( \Delta_i \), and is defined by,

\[ \phi(r) = \begin{cases} 2 \min(1, r) / (1 + r), & r > 0 \\ 0, & \text{otherwise} \end{cases} \]  
(2.26)

where \( r \) is the ratio of slopes on each side of the cell,

\[ r_k = \frac{(\Delta \mathbf{U}_{i-\frac{1}{2}})_k}{(\Delta \mathbf{U}_{i+\frac{1}{2}})_k}, \]  
(2.27)

is the van Leer limiter, used for this study. Figure 2.3 shows Equation 2.27 plotted along with the permissible region for a slope limiter which does not introduce oscillations itself. As shown in Figure 2.2, the left and right interpolated values from Equation 2.24 are then advanced by a half time step, \( \Delta t/2 \),

\[ \mathbf{\bar{U}}_i^L = \mathbf{U}_i^L + \frac{\Delta t}{2 \Delta x} [\mathbf{F}((\mathbf{U}_i^L) - \mathbf{F}(\mathbf{U}_i^R)], \]
\[ \mathbf{\bar{U}}_i^R = \mathbf{U}_i^R + \frac{\Delta t}{2 \Delta x} [\mathbf{F}(\mathbf{U}_i^L) - \mathbf{F}(\mathbf{U}_i^R)]. \]  
(2.28)

These advanced values are then used as the input for a Riemann solver at each cell interface, with the \( \mathbf{\bar{U}}_i^R \) and \( \mathbf{\bar{U}}_{i+1}^L \) as the left and right states at the \( \mathbf{\bar{U}}_{i+\frac{1}{2}} \) boundary, allowing for the solution to be advanced by combining with Equation 2.18,

\[ \mathbf{U}_{i+1}^{n+1} = \mathbf{U}_i^{n} + \frac{\Delta t}{\Delta x} \left[ \mathcal{F}(\mathbf{U}_{i+1}^L, \mathbf{\bar{U}}_i^R) - \mathcal{F}(\mathbf{\bar{U}}_i^L, \mathbf{U}_{i+1}^R) \right], \]  
(2.29)

where \( \mathcal{F} \) denotes the flux computed from solving the Riemann problem with given left and right states. To ensure positivity of the reconstructed pressure, we reconstruct the primitive state vector \( \mathbf{q} = (\rho, \mathbf{u}, p) \) rather than the conservative state vector \( \mathbf{U} = (\rho, \rho \mathbf{u}, E) \).
2.4.3 Higher-order Non-MUSCL methods

The MUSCL–Hancock method has the appealing quality of achieving second-order convergence in time with relatively little computational expense. However, for the irregular grid problems introduced further into this work, the implementation of such a centred scheme is more challenging. Instead, we return to Equation 2.24, having reconstructed states at the left and right boundaries of each cell. To achieve higher order convergence in time\(^2\), we turn to a Runge–Kutta integrator.

Provided that our discretisation in space has been performed independently of time (the rather fancifully named “method of lines” technique), such that we wish to solve,

\[
\frac{d\mathbf{q}_i}{dt} = \mathcal{L}(\mathbf{q}_i),
\]

\(^2\)And indeed this turns out to be a necessity; with a first-order time integrator, a Godunov scheme based on linear reconstruction is non-TVD.
then we employ the strongly stability preserving integrator,

\[
q_i^{(1)} = q_i^n + \Delta t \mathcal{L}(q_i),
\]

\[
q_i^{(2)} = q_i^{(1)} + \Delta t \mathcal{L}(q_i^{(1)}),
\]

\[
q_{i+1}^{n+1} = \frac{1}{2} (q_i^n + q_i^{(2)}).
\]

Note that this is equivalent to performing two forward Euler time integration steps each of length \(\Delta t\), and then averaging with the initial data. Thus we recover a centred scheme again, in the sense that the time \(t + \Delta t\) is half way between the times \(t\) and \(t + 2\Delta t\) from which data are drawn.

2.4.4 CFL CONDITION

This flux update formula is stable for time steps which satisfy the inequality (Courant et al., 1928),

\[
\frac{\Delta t}{\Delta x} \max_i(S) < 1,
\]

where \(\max_i(S)\) denotes the maximum wave speed on the grid. Intuitively, this means that a signal cannot propagate more than one cell in a single time step. Rearranged, this provides a formula for \(\Delta t\) in terms of the CFL number, \(C_{CFL}\),

\[
\Delta t = C_{CFL} \frac{\Delta x}{\max_i(S)}.
\]

The stability criterion easily generalises to multiple dimensions,

\[
\Delta t = C_{CFL} \min \left( \frac{\Delta x}{\max_{i,j,...}(S_x)}, \frac{\Delta y}{\max_{i,j,...}(S_y)}, \ldots \right),
\]

where \(S_x\) is the maximum wave speed in the \(x\) direction, and \(\max\) runs over all dimensions. The CFL number is taken to be 0.9 throughout this report unless otherwise stated, although it is reduced to 0.2 for the first five time steps following the advice in Toro (1999).
Chapter 3

CFD on the GPU

In recent years, general purpose graphics processing units (GPGPU) have become an affordable and practical means of performing numerical simulations. As of May 2013, high-end consumer units produced by Nvidia exceed a computation rate of 3000 GFLOPS (floating point operations per second) for a retail price of $400 and power consumption of 250 W, compared with \( \sim 40 \) GFLOPS for a current Intel i7 processor of a similar price and 50 W power consumption. Thus, if the full performance of the GPU can be used, it offers roughly \( 15\times \) the performance per unit price per unit power.

GPGPUs have already been used to accelerate a variety of problems in scientific computing. Wang et al. (2010) reports the implementation of an adaptive mesh refinement (AMR) code for astrophysical simulations on a cluster of Nvidia GPUs with \( 10\times \) speed-up on a single GPU over a single CPU code, as well as near-ideal parallelisation over multiple computers. Similar success is described in Maitre et al. (2009), with a \( 30\times \) speed-up for an evolutionary algorithm for condensed matter physics.

Previous research within the laboratory (Lovett, 2009) has also shown the feasibility of using CUDA to accelerate Riemann problem based fluids solvers. Previous experience by the author has noted a \( 100\times \) performance improvement for simple problems such as ideal gas simulation on a fully regular Cartesian grid, and \( 25\times \) for a complex multi-material model with iterative relaxation steps\(^1\). In general, the benefits of using CUDA depend strongly on the characteristic of the algorithm.

\(^1\)The non-parallelised model is now published as Schoch et al. (2013).
3.1 Processing architecture

This performance does not come without caveats. The processing structure of a graphics card is optimised heavily for processing images, which generally involves applying the same operation to each component (pixel) of an image; this is known as stream processing, related to the single instruction, multiple data (SIMD) paradigm. A stream of data is read in, and the same computational operations (instructions) are applied to each data point in the stream. CPUs are generally scalar processors, acting with a single instruction on a single piece of data (SISD). Most of the transistors on a modern CPU are devoted to the L1 and L2 caches which reduce the latency incurred by loading the next piece of data from memory. By contrast, applying the same set of operations to a larger block of data allows a GPU to hide the latency created by a slower memory controller and smaller cache.

3.2 Suitable problems

Not all problems are easily translated to run efficiently on a GPU. Three key requirements must be fulfilled:

- **Parallelisability**: foremost, the problem under consideration must be parallelisable. That is to say, it must be possible to separate the problem into a number of parallel tasks. The most parallelisable problems (often termed embarrassingly parallel) involve no communication between these parallel tasks. An example of such a problem is computing points in the complex plane which lie within the Mandelbrot set, or brute-force searches in cryptography. On the other end of the spectrum, problems like parsing a computer program are generally not parallelisable at all, as they involve an inherently ordered set of tasks.

- **Locality of data**: the poor latency of Nvidia’s GPU architecture (equivalent to as many as 400–600 cycles) means that memory access between different parallel tasks should be in a technical sense local. Accessing arbitrary memory locations which, for example, depend on the outcome of previous computations can have very poor performance; an example of this is computing streamlines by direct integration.

- **Computationally intensive**: due to poor memory performance, tasks must be computationally intensive, meaning that most of the work involves computation rather than memory manipulation. Algorithms for problems such as database manipulations are typically memory intensive, and so not generally suitable for
GPU parallelisation.

The solution of hyperbolic partial differential equations is an extremely good fit for these criteria. Data locality is ensured by the hyperbolic nature of the system; each cell has a region of causal contact at each time step, typically one or two cells on either side depending on the order of the method. The solution method is also identical for each cell on a Cartesian grid, so the problem is trivially parallelisable; the only significantly non-ideal part of the algorithm is the Riemann solver itself, which generally follows a number of different branches.

3.3 **Compute Unified Device Architecture (CUDA)**

In this case, the entire solver is implemented using CUDA, so as to run on Nvidia general purpose graphics processing units (GPGPUs). With CUDA, the solution for each cell is performed by a single *thread*, with threads being logically grouped into *blocks* of around 64 threads which have capability for inter-thread communication. Up to 512 threads are then executed in parallel on the GPU. By executing multiple blocks at the same time, the GPU is able to “swap” between blocks in order to hide memory latency and maximise time spent performing calculations.

![Figure 3.1: The thread and block structure of the implementation of a Riemann solver with a one cell overlap between blocks.](image)

Due to the large stencil of second order schemes (as shown in Figure 2.4), an overlapping cell at the end of each block is required in the solution direction, as shown in Figure 3.1. Each cell loads data $U_{i-1}$, $U_i$, and $U_{i+1}$, and reconstructs $U_i^L$ and $U_i^R$. Then, the $i^{th}$ cell computes the flux $F_{i-\frac{1}{2}}$ using the reconstructed values from cells $i-1$ and $i$. The overlap region is required as this flux cannot be computed in the first cell in a block as there is no cell to the left of it.
The number of blocks required to cover a grid with a given overlap at each block boundary is given by,

\[
\text{blocks} = 1 + \left\lceil \frac{\text{grid cells}}{\text{block width} - \text{overlap}} \right\rceil. \tag{3.1}
\]
One approach to representing immersed boundaries is the ghost fluid method, proposed in Fedkiw et al. (1999). The original method allows for discretisation of the domain on both sides of the boundary with the material on each side described by a different equation of state. An extrapolation procedure in density, velocity, and entropy from each side to the other is used to update a narrow band of ghost cells to allow for a single-material solver to be used for each fluid. This allows for significantly simpler implementation than an explicit multi-material Riemann solver. For our purposes, the method is easily adapted to represent solid objects by discretising the domain on only one side of the boundary.

Each conserved variable $I$ is extrapolated into the ghost region using,

$$I_\tau + \hat{n} \cdot \nabla I = 0,$$

where $\tau$ is a fictitious time, and $\hat{n}$ is the gradient of the signed distance function,

$$\hat{n} = \nabla \phi = (\phi_x, \phi_y)^T.$$

### 4.1 Signed distance function

At each computational cell, we define a scalar function $\phi(x)$ (the signed distance function) which encodes the shortest distance to the solid boundary. The contour $\phi = 0$ is the boundary, and we choose the convention that $\phi > 0$ denotes the interior and $\phi < 0$ the exterior of the solid object. This signed distance function can be obtained
analytically in many simple cases (such as spheres, cuboids, and general polyhedra), while more general cases can be handled by importing triangulated meshes.

A true signed distance function should obey $|\nabla \phi| = 1$ almost everywhere, but for numerical purposes, $\hat{n}$ should be normalised whenever computed. As Equation 4.1 is solved to steady-state, only first-order approximations to $I_x$ and $I_y$ are necessary. To ensure that information propagates from the real fluid to the ghost cells, the derivatives are approximated as,

$$\frac{\partial I_{n,m}}{\partial x} \approx \frac{1}{\Delta x} \begin{cases} (I_{n,m} - I_{n-1,m}), & \phi_x > 0 \\ (I_{n+1,m} - I_{n,m}), & \phi_x < 0 \end{cases},$$

and likewise for $y$. Equation 4.1 discretises to,

$$I^{k+1} = I^k - \Delta \tau (\phi_x n_x + \phi_y n_y).$$

(4.4)

Iterating 20 times with $\Delta \tau = \Delta x / 2$ is enough to initialise a band of ghost cells in the range $0 \leq \phi \leq 3\Delta x$. To create a reflective boundary, the component of the velocity normal to $\hat{n}$ must be reversed,

$$u' = u - 2(u \cdot \hat{n})\hat{n}.$$  

(4.5)

However, while Equation 4.1 is an explicitly two dimensional equation, the boundary is only reconstructed with granularity at the level of computational cells (somewhat of a zeroth order approximation, in that the actual boundary length is non-convergent). As such, waves propagating along the boundary encounter a staggered grid, which can lead to significant numerical artefacting. While the ideal solution is a higher order representation of the boundary, the problem can be somewhat ameliorated by using additional information from the geometry. In addition to $\phi(x)$, the fraction of the cell covered by the boundary $\lambda \in [0,1]$ is also stored. Equation 4.5 is then modified to reflect the velocity in a way proportional to the cell coverage,

$$u' = u - 2\lambda(u \cdot \hat{n})\hat{n}.$$  

(4.6)

This equation can be modified to obtain the extrapolated velocity from a boundary moving with velocity $v$ by adding an additional component of velocity proportional to the projection of $v$ onto the boundary normal $\hat{n}$,

$$u' = u - 2\lambda(u \cdot \hat{n})\hat{n} + 2\lambda(v \cdot \hat{n})\hat{n}. $$

(4.7)

With a boundary moving across the grid, a further complication arises. As the level set function changes, there will be cells for which $\phi_{i,j} > 0$ and $\phi_{i,j+1} < 0$ or vice versa; that
**Ghost fluid method**

is cells which are uncovered or covered in a time step. Failing to address this problem specifically will lead to poorly extrapolated values which have had the velocity normal to the boundary reversed. As such, the algorithm is modified to extrapolate values across the boundary using Equation 4.4 *without* applying Equation 4.7. The level set is then reinitialised in the new position, and the relaxation procedure applied *with* velocity reflection.

Integrals over the interior or surface of the level set can be written as a sum including a term dependent on $\phi$. For example, a surface integral to find the force exerted on the boundary due to imbalance in pressure can be written and discretised as,

$$ F = \int_{\partial V} p \, dS, $$

$$ = \int_V p \, \hat{n} \, \delta(\phi) \, dV, $$

$$ \approx \sum_{i=0}^{N} \sum_{j=0}^{M} p_{i,j} \, \hat{n} \, \delta_s(\phi_{i,j}), $$

(4.8)

where $\delta(x)$ is the Dirac delta function, and $\delta_s(x)$ is a smoothed version defined as,

$$ \delta_s(x) = \frac{1}{\Delta x \sqrt{2\pi}} \exp \left( -\frac{x^2}{2\Delta x^2} \right), $$

(4.9)

which allows for evaluation on a discrete grid. Attempting to use $\delta(\phi(x))$ directly would give erroneous results, as aside from trivial cases involving simple grid-aligned boundaries, one would not expect any cell to have exactly $\phi = 0$. Convergence of Equation 4.8 is guaranteed as $\delta_s \rightarrow \delta$ as $\Delta x \rightarrow 0$.

**4.2 Results**

Results for a number of test cases are presented using the ghost fluid method for comparison to those obtained using the $h$-box cut cell method. This link will be made more clear in the final thesis.

**4.2.1 Static cylinder**

The passage of a shock around a cylinder of radius 0.2 m centred at (0.6, 0.5) m is simulated at multiple resolutions. Upon initial impact of the shock with the curved front face of the cylinder, some spurious oscillations occur, as depicted in Figure 4.1.
Results

These are clearly visible in the density schlieren\textsuperscript{1} plot, but less so in the density plot, and are quickly diffused so as to not be observable at later times.

Figure 4.2 shows visualisation of several primitive variables of the flow along with density schlieren images at selected times. Regular reflection of the incident shock begins at $t = 0.15$ s. However, as the shock traverses regions of the boundary with more shallow gradient, Mach reflection occurs. The three shock waves of Mach reflection are visible at $t = 0.25$ s, while the slip plane is more apparent at the final time pictured ($t = 0.4$ s).

Slices through the domain at the final time are shown in Figure 4.3, in which the reflected shock, Mach stem, and slip plane are all visible, the last being seen as a small discontinuity at around $x = 0.65$ m, and only clearly resolved at high resolution.

![Figure 4.1: Top: density and bottom: density schlieren images just after the initial contact of the shock with the curved surface of the cylinder at $t = 0.15$ s.](image)

\textsuperscript{1}Schlieren images are imitated with the function, 

$$\exp(-\kappa \nabla \rho),$$

where $\kappa$ is a small positive constant, roughly proportional to $\max(\nabla \rho)$, selected to highlight features.
Figure 4.2: 2D visualisation of the shock–rigid cylinder impact case. Variables are: top row, density (1 kg m\(^{-3}\) to 2 kg m\(^{-3}\)); second row, \(x\) velocity (0 m s\(^{-1}\) to 0.75 m s\(^{-1}\)); third row, \(y\) velocity (−0.25 m s\(^{-1}\) to 0.25 m s\(^{-1}\)); bottom row, density schlieren. Output times, from left, \(t = (0.15, 0.25, 0.4)\) s.
Results

Figure 4.3: Data taken along the line $y = 0.275\,\text{m}$ at $t = 0.4\,\text{s}$ for the shock–rigid cylinder impact case. Results are presented at three different resolutions to illustrate the scale-dependent behaviour of the solution.
4.2.2 Steady-state aerofoil flow

The steady-state flow around an aerofoil is simulated. The symmetric NACA aerofoil is defined by the equation,

\[ y(x) = 5tc \left[ 0.2969 \sqrt{\frac{x}{c}} - 0.1260 \left( \frac{x}{c} \right) - 0.3516 \left( \frac{x}{c} \right)^2 + 0.2843 \left( \frac{x}{c} \right)^3 - 0.1036 \left( \frac{x}{c} \right)^4 \right] \]

(4.10)

where \( c \) is the chord length, and \( t \) is the maximum thickness relative to the chord length. In a four digit aerofoil, \( 100t \) is the last two digits of the identifying code. For example, the NACA 0012 aerofoil depicted in Figure 4.4 has a maximum thickness of 12% of the chord length.

Figure 4.4: NACA 0012 airfoil.

While the Riemann solvers described in this thesis are not commonly used for steady-state aerospace flow computation, simulation of the flow around an aerofoil was attempted. The aerofoil is represented by a fixed level set, so the signed distance function can be (somewhat inefficiently) obtained by explicit minimisation of \((x' - x)^2 + (y' \pm y(x))^2\) with respect to \( x \in [0, c] \) for each point \((x', y')\) on the grid. Boundary conditions for the edges of the domain are more complex; for subsonic inflow, we may only specify two variables at the boundary. Data are copied into ghost cells in the normal manner, then entropy \( s = p/\rho^\gamma \) and \( x\)-momentum are specified based on the flow values at infinity. For the upstream boundary, a constant pressure outflow condition is applied.

The pressure coefficient is defined as,

\[ C_p = \frac{p - p_\infty}{\frac{1}{2} \rho_\infty v_\infty^2}, \]

(4.11)

where \( p_\infty, \rho_\infty, \) and \( v_\infty \) are the values of pressure, density, and velocity of the flow far from the aerofoil. Values of the pressure coefficient along the contour of a NACA 0012 aerofoil in Mach 0.4 flow are shown in Figure 4.5. Reasonable agreement with the validation results from Leese (2010) is seen, in which similar problems occur in capturing the flow at the rear of the aerofoil. In this region, the ghost fluid approach is broken by the inability to accurately compute a normal across regions of width \( \lesssim 2\Delta x \).

\footnote{To paraphrase Phil Roe, boundary conditions are a mathematical model for the rest of the universe.}
Figure 4.5: Pressure coefficient around a NACA 0012 aerofoil at zero angle of attack in Mach 0.4 flow, simulated to steady state using the ghost fluid method.

Figure 4.6 shows the steady-state simulation of Mach 0.8 flow around the same aerofoil geometry, performed on the domain $[-2, 3] \times [-1.5, 1.5]$m discretised with $2000 \times 1200$ cells and the aerofoil in the centre. Severe numerical errors appear around the front of the aerofoil where the incident flow is accelerated past the ragged boundary. This simulation did not converge to a correct solution with higher resolution, and the errors at the boundary spread into the bulk of the fluid as well.

Figure 4.6: Pressure coefficient around a NACA 0012 aerofoil at zero angle of attack in Mach 0.8 flow, simulated to steady state using the ghost fluid method.

While this result, combined with the oscillations produced at the leading edge of
a cylinder impacted by a shock wave (shown in Figure 4.1) may seem damning of the ghost fluid method, they must be critically viewed with some caveats. The ghost fluid algorithm used in this work does not represent the state-of-the-art in ghost fluid-like algorithms. Work by Sambasivan and UdayKumar (2009) and Tan et al. (2012) show that improvements to the boundary solution can be made within the same framework. Unfortunately, there are dozens of similar algorithms, and it is beyond the scope of this work to test them all.

4.2.3 Mass loss

To test the mass conservation properties of the ghost fluid method, we use a simplified test case which will exhibit the same problems as a more complex fracture simulation, while being easier to reproduce. The simulation is performed with uniform initial conditions, \( p = 1 \text{ Pa}, \rho = 1 \text{ kg}, \) and \( \mathbf{v} = 0 \). The fluid is an ideal gas with \( \gamma = 1.4 \). The domain consists of a \([-1 \text{ m}, 1 \text{ m}] \otimes [-1 \text{ m}, 1 \text{ m}]\) with reflective boundaries. A cylinder of radius 0.5 m is initialised centred at (0.5 m, 0) and rotates with angular frequency \( \omega = \pi/5 \text{ s}^{-1} \).

![Figure 4.7: Mass error (\( \Delta m \)) as a function of time for the test case described in the text. Traces are shown at multiple resolutions. Past \( t \approx 0.1 \) cycles, the error monotonically decreases with resolution.](image)

Given that all the boundaries are reflective, the exact solution should conserve mass exactly. Using the ghost fluid method with a fractional volume coverage, the total mass is given by,

\[
m = \sum_{i,j} \lambda_{i,j} \rho_{i,j} \Delta x \Delta y.
\]
Results

The exact mass is $4 - \pi/16 \cong 3.804$. Figure 4.7 shows the deviation from the exact mass over the course of two full rotations of the cylinder. After an initial dip at start-up, the mass steadily rises throughout the simulation. The rate of mass accretion ($\frac{d\Delta m}{dt}$) decreases with increasing resolution, indicating convergence to a conservative scheme. Table 4.1 shows the fractional mass error after one cycle for a range of resolutions, with an estimate of the convergence rate (to $\Delta m = 0$). If $e_1$ and $e_2$ denote the errors in successive simulations with cell spacing $\Delta x_1$ and $\Delta x_2$ respectively, the order of convergence can be estimated as,

$$k = \frac{\log(e_2) - \log(e_1)}{\log(\Delta x_2) - \log(\Delta x_1)}.$$  \hspace{1cm} (4.13)

<table>
<thead>
<tr>
<th>Resolution</th>
<th>$\Delta m$/kg after 1 cycle</th>
<th>Fractional error</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 x 100</td>
<td>0.0678</td>
<td>1.78%</td>
<td>—</td>
</tr>
<tr>
<td>200 x 200</td>
<td>0.0378</td>
<td>0.99%</td>
<td>0.84</td>
</tr>
<tr>
<td>400 x 400</td>
<td>0.0201</td>
<td>0.53%</td>
<td>0.91</td>
</tr>
<tr>
<td>800 x 800</td>
<td>0.0104</td>
<td>0.27%</td>
<td>0.95</td>
</tr>
</tbody>
</table>

Table 4.1: Mass error for the test case described in the text after one complete cycle. The convergence rate between sequential resolutions is approximated using Equation 4.13.
The ghost fluid method represents complex boundaries by adding additional parameters to a Cartesian grid without altering its structure; the methods described in Chapter 4 detail the extent to which the algorithm can be improved to suppress numerical artefacts. To achieve a higher order representation of the boundary, Cartesian cut cell methods construct a computational grid from the intersection of a regular Cartesian grid with a complex geometry. This approach lies between fully Cartesian and unstructured grids in terms of grid complexity. The regular order of the Cartesian mesh is retained throughout most of the domain, allowing for the use of efficient finite volume solvers, while accurately representing the complex geometry through a single layer of irregular cells, as shown in Figure 5.1.

Such meshes are trivial to produce. The ghost fluid method requires that a signed distance function be initialised throughout the entire computational domain, whereas to compute an embedded boundary, only the intersections of the boundary with computational cells are required. For example, creating a signed distance function from a triangulated mesh is computationally expensive for large grids, while computing the intersection of facets with grid lines or planes is efficient.

Attention must then be turned to how the numerical solution should be advanced in the irregular cells. The most obvious approach might be to use a conventional irregular grid finite volume solver for the cut cells. However, from Equation 2.33, the stable time step is proportional to the length scale of the smallest cell. As the irregular cells can be many orders of magnitude smaller than regular cells, this leads to an extremely small
time step restriction which dramatically increases computational effort and numerical diffusion.

There are several approaches to dealing with the small cell problem:

1. **Cell merging:** Described in Ingram et al. (2003). Small cells of area fraction $\alpha$ can be merged with adjacent uncut cells to form a larger cell of area fraction $1 + \alpha$ which then no longer limits the time step. This technique has several caveats; firstly, the resolution of the grid is markedly reduced for cut cells, and secondly, finding an appropriate cell with which to merge can be difficult in 2D and higher dimensions.

2. **Flux-stabilisation method:** An explicitly dimensionally split method described in Klein et al. (2009) which stabilises the flux update in cut cells by splitting each edge into regions which are shielded and unshielded by the cut boundary edge. The scheme is conservative and first-order near boundaries. However, it has the significant advantage of being possible to implement in a fully split scheme for computational efficiency.

3. **Mesh reshaping:** Developed in Sachdev and Groth (2007), this method involves moving the previously grid-aligned cut cell edges to eliminate tiny cells around the boundary. This allows for accurate capturing of the boundary (including sharp edges which might be blunted by other methods), but introduces a rather odious mesh generation procedure.

4. **h-box method:** A novel procedure suggested in Berger and Helzel (2012) involves the construction of quadrilaterals on either side of each cut edge which encompass the domain of dependence of the flux calculation at that edge (that

Figure 5.1: A low-resolution sample mesh used for a cut cell method, created from the intersection of a circle of unit radius with a $[-1.5, 1.5] \times [-1.5, 1.5]$ domain discretised into $31 \times 31$ cells. Most of the domain is described by a square Cartesian grid, with irregular polygonal cells in the vicinity of boundaries.
Cut cell methods

is, have length \( \propto \Delta x \). A rotated grid Riemann solver proposed in Helzel et al. (2005) is then used, with characteristic directions chosen to fulfil a cancellation property. The method is both conservative and second-order at boundaries.

This thesis focuses on the \( h \)-box method.

<table>
<thead>
<tr>
<th></th>
<th>( \mathcal{O}(\Delta x^2) )</th>
<th>Moving 3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell merging</td>
<td>( \times )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>Cell shielding</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Mesh reshaping</td>
<td>( ?^\dagger )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>( h )-box</td>
<td>( \checkmark )</td>
<td>( \times )</td>
</tr>
</tbody>
</table>

\( ^\dagger \) Not tested at the boundary.

Table 5.1: \( \checkmark \) denotes published results, \( \times \) their absence.

5.1 Notation

In order to describe the flow through cut cells, it is useful to begin with the rigorous definition of a number of quantities. Let the flow be computed on a rectangular domain of finite extent,

\[
D = \{ x, y : x_{min} \leq x \leq x_{max} \land y_{min} \leq y \leq y_{max} \},
\]

(5.1)

which is discretised by a \( N \times M \) grid such that the size of each computational cell is \( \Delta x \times \Delta y \) where,

\[
\Delta x = \frac{x_{max} - x_{min}}{N} \quad \text{and} \quad \Delta y = \frac{y_{max} - y_{min}}{M}.
\]

(5.2)

Each cell is indexed by two integers \( i \in [0, N), \; j \in [0, M), \)

\[
C_{i,j} = \{ x, y : x_{i-\frac{1}{2}} \leq x \leq x_{i+\frac{1}{2}} \land y_{j-\frac{1}{2}} \leq y \leq y_{j+\frac{1}{2}} \} \]

(5.3)

The cell \( C_{i,j} \) has centroid,

\[
\mathbf{x}_{i,j} = \begin{pmatrix} x_i \\ y_j \end{pmatrix}.
\]

(5.4)

We now introduce a subspace of the computational domain Equation 5.1 which corresponds to a solid obstruction \( S \) (the “dry” region). Let \( V = D \setminus S \) be the unobstructed subspace of \( D \) (the “wet” region) through which is occupied by fluid. We define cut cells as the intersection of the wet region with each computational cell,

\[
\tilde{C}_{i,j} = C_{i,j} \cap V.
\]

(5.5)
We now define functions corresponding to the area and centroid of the finite Euclidean space $\Omega$,

$$\alpha(\Omega) = \int_{\Omega} dx \, dy,$$

$$\mathbf{x}(\Omega) = \frac{1}{\alpha(\Omega)} \int_{\Omega} \mathbf{x} \, dx \, dy,$$

(5.6) (5.7)

with short-hand notation for regular grid cells and cut cells,

$$\alpha_{i,j} = \alpha(C_{i,j}), \quad \tilde{\alpha}_{i,j} = \alpha(C_{i,j} \cap V)$$

$$\mathbf{x}_{i,j} = \mathbf{x}(C_{i,j}), \quad \tilde{\mathbf{x}}_{i,j} = \mathbf{x}(C_{i,j} \cap V).$$

(5.8) (5.9)

The Cartesian boundaries of computational cells are defined as,

$$\partial C_{i \pm \frac{1}{2},j} = \{x, y : x = x_{i \pm \frac{1}{2}} \land y_{j-\frac{1}{2}} \leq y \leq y_{j+\frac{1}{2}}\},$$

$$\partial C_{i,j \pm \frac{1}{2}} = \{x, y : x_{i-\frac{1}{2}} \leq x \leq x_{i+\frac{1}{2}} \land y = y_{j \pm \frac{1}{2}}\},$$

(5.10) (5.11)

and the corresponding cut cell boundaries as,

$$\tilde{\partial} C_{i \pm \frac{1}{2},j} = \partial C_{i \pm \frac{1}{2},j} \cap V.$$

(5.12)

A fifth edge is created by the intersection of the boundary of $V$ with cells through which it passes,

$$\tilde{\partial} C_{i,j} = C_{i,j} \cap \partial S.$$

(5.13)

The lengths of these edges are denoted by,

$$\tilde{\ell}(\Gamma) = \int_{\Gamma} dl,$$

(5.14)

with corresponding short-hands $\ell_{i \pm \frac{1}{2},j}$ for edges of regular cells, and $\tilde{\ell}_{i \pm \frac{1}{2},j}$ for edges of cut cells.

5.2 Description of geometry

Cut cell methods in general do not allow for an arbitrary description of the surface. Instead, the boundary is approximated by a polygon whose vertices all lie on cell edges. This results in a piecewise linear reconstruction of the boundary in cut cells. Additionally, we impose the restriction that each cut cell be connected; that is, the cut edge partitions the cell into only two regions. Figure 5.2 shows an example in which a sharp edge leads to the creation of a cut cell with two triangular disconnected regions; this is resolved by removing any boundary–edge intersections which share the
same edge. Quirk (1994) follows a similar approach, and notes that this leads to the blunting of sharp features. However, he combines the cut cell algorithm with Adaptive Mesh Refinement, so that the rounding effect can be reduced to arbitrarily small scales. Quirk (1994) also raises concerns with the use of finite precision arithmetic to compute boundary–grid intersections. Rounding errors using floating point arithmetic can lead to mistakes in mesh generation, which can be either minor (a very slightly incorrect edge length) or fatal (an edge intersection completely ignored).

5.3 The \( h \)-box method

Berger and Helzel (2012) describes a simplified \( h \)-box method which avoids the wave-propagation formulation from Berger et al. (2005). For edges surrounding cut cells, the grid is projected on to a quadrilateral grid with axis vectors generally normal and parallel to the solid boundary with similar grid spacing. Conventional Riemann-problem based methods can then be applied to the rotated grid, and the fluxes projected back. Once the fluxes have been computed, the complete flux update formula is,

\[
U_{i,j}^{n+1} = U_{i,j}^{n} - \frac{\Delta t}{\alpha_{i,j}} \left[ (\ell F)_{i+\frac{1}{2},j}^{n+\frac{1}{2}} - (\ell F)_{i-\frac{1}{2},j}^{n+\frac{1}{2}} + (\ell F)_{i,j+\frac{1}{2}}^{n+\frac{1}{2}} - (\ell F)_{i,j-\frac{1}{2}}^{n+\frac{1}{2}} - (\ell F)_{i,j}^{n+\frac{1}{2}} \right],
\]

(5.15)

where \( \ell_{i+\frac{1}{2},j} \) and \( \ell_{i,j+\frac{1}{2}} \) are the lengths of the Cartesian grid edges, which may be shorter than \( \Delta x \) due to the cut, or even exactly zero if the edge is fully inside the boundary; and \( \ell_{i,j} \) is the length of the non-Cartesian edge formed by the boundary. The fractional time index for each flux \( F \) indicates the flux between times \( n \) and \( n + 1 \). The flux \( F_{i,j} \) can be deduced without solving a Riemann problem; as the boundary is solid and stationary, there is zero mass flux, and the work done by the wall is \( dW = F \cdot dx = 0 \). However, momentum of the fluid is not conserved, as a parcel of gas reflected by the
solid boundary would have its momentum reversed. The total flux function is given by,
\[
F_{n+rac{1}{2}} = \begin{pmatrix} 0 \\ p^* \hat{n} \\ 0 \end{pmatrix}.
\] (5.16)

5.3.1 Non-orthogonal rotated grid method

A key component of the \( h \)-box method is the computation of fluxes at cell edges by combining fluxes calculated in directions non-orthogonal to the grid. This section provides a brief summary of the method described in Helzel et al. (2005). Taking two general directions \( \eta = (\eta_x, \eta_y)^T \) and \( \xi = (\xi_x, \xi_y)^T \), the flux in the \( \eta \) and \( \xi \) directions can be expressed as,
\[
f_\eta = \eta_x f_x + \eta_y f_y,
\] (5.17)
\[
f_\xi = \xi_x f_x + \xi_y f_y,
\] (5.18)
where \( f_x \) and \( f_y \) are the flux functions in the \( x \) and \( y \) directions respectively. Note that for spatially homogeneous problems (such as the Euler equations), the two flux functions are trivially related by a rotation matrix applied to the velocity component,
\[
f_y = R\left(-\frac{\pi}{2}\right)f_x R\left(\frac{\pi}{2}\right).
\] (5.19)

The fluxes in the \( \eta \) and \( \xi \) directions must then be combined to provide a flux in the \( x \) direction through a cell edge. The flux through the edge \( (i - \frac{1}{2}, j) \) is given by,
\[
F_{i-\frac{1}{2},j} = A F_{\eta i-\frac{1}{2},j} + B F_{\xi i-\frac{1}{2},j},
\] (5.20)
which, for consistency with \( f_x \) must satisfy,
\[
A (\eta_x f_x + \eta_y f_y) + B (\xi_x f_x + \xi_y f_y) = f_x.
\] (5.21)

This provides a small linear system of equations to determine the unknown coefficients \( A \) and \( B \),
\[
\begin{pmatrix} \eta_x & \xi_x \\ \eta_y & \xi_y \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix},
\] (5.22)
which has the solution,
\[
A = \frac{-\xi_y}{\xi_x \eta_y - \xi_y \eta_x}, \quad B = \frac{\eta_y}{\xi_x \eta_y - \xi_y \eta_x}.
\] (5.23)

The procedure for the flux \( F_{i,j+\frac{1}{2}} \) is analogous, and not repeated for brevity. From Equation 5.23, obviously \( \eta \) and \( \xi \) must not be parallel (i.e., must form a basis of
Berger and Helzel (2012) describes the use of an orthogonal rotated grid method used for cut cells at smoothly varying boundaries; this corresponds to the special case \( \eta \cdot \xi = 0 \).

### 5.3.2 Construction of \( h \)-boxes

At smoothly varying boundaries, the rotated grid method can be applied with \( \eta = \hat{n} \), the boundary normal, and \( \xi \), a direction perpendicular to it. This leads to the question of how to obtain the state vector \( q^L_\eta \) and \( q^R_\eta \). Consider a triangular cut cell bounded by a linear boundary segment as shown in Figure 5.3. Figure 5.3 (a) shows a box \( q^R \) of length \( h \) constructed from the boundary segment in the normal direction into the flow domain, overlapping four grid cells. The value in this box is obtained by performing an integral over piecewise linear reconstructed values of the primitive variables of the grid. Figure 5.3 (b) shows the two normal \( h \)-boxes constructed at the two Cartesian edges of the cut cell. Note that the difference between boxes on the left and right is the area of the cut cell.
The \( h \)-box method

size of the cut cell. This is the “cancellation property” which allows the method to be stable; the fluxes cancel in the \( \eta \) direction to introduce a factor of \( \alpha \) in the numerator of the flux update to cancel that in the denominator. Figure 5.3 (c) and (d) show the construction of \( h \)-boxes in the \( \xi \) direction from the edges \((i - \frac{1}{2}, j)\) and \((i, j - \frac{1}{2})\); again, the difference is the area of the small cell.

The length of the \( h \)-box is determined by the scheme used. For an unsplit scheme, which is stable provided,

\[
\max_{i,j} (|u_x| + |u_y|) \frac{\Delta t}{\Delta x} \leq 1,
\]

the suggested \( h \)-box length is,

\[
h = \sqrt{\eta_x^2 + \eta_y^2} \frac{1}{|\eta_x| + |\eta_y|}.
\]

Values in \( h \)-boxes are calculated using an area integral over the reconstructed values on the underlying grid. As we employ a piecewise linear reconstruction, this reduces to an area-weighted sum of values at the centroids of the intersection of the \( h \)-box with each computational cell,

\[
q_{h\text{-box}} = \sum_{i,j} \frac{\alpha(\tilde{C}_{i,j} \cap H) \{ q_{i,j} + [x(\tilde{C}_{i,j} \cap H) - x(\tilde{C}_{i,j})] \cdot \nabla q_{i,j} \}}{\sum_{i,j} \alpha(\tilde{C}_{i,j} \cap H)},
\]

where \( C_{i,j} \cap H \) denotes the intersection of the \((i, j)^{th}\) Cartesian grid cell with the \( h \)-box, \( A_k \) the area of a polygon, and \( x_k \) its centroid; \( q_{i,j} \) is the value and \( \nabla q_{i,j} \) the limited gradient in the \((i, j)^{th}\) cell. The gradient in the \( h \)-box is also estimated by an area-weighted averaging,

\[
\frac{dq}{d\xi_{h\text{-box}}} = \sum_{i,j} \frac{\alpha(\tilde{C}_{i,j} \cap H) \{ (\nabla q)_{i,j} \cdot \xi \}}{\sum_{i,j} \alpha(\tilde{C}_{i,j} \cap H)},
\]

and likewise for \( \xi \).

Having been reconstructed, the velocities must be rotated into the \((\eta, \xi)\) coordinate system. Defining \( L_{ij} \) as the rotation matrix transforming\(^1\) from \( \mathbf{x} = (x, y)^T \) to \( \mathbf{x}' = (\eta, \xi)^T \) such that \( x_i' = L_{ij}x_j \), the velocities transform as,

\[
\mathbf{v}' = L\mathbf{v}.
\]

The gradient of velocity is more complex; the gradient of a vector field is a matrix with rows corresponding to the gradients of each component of the vector and transform

\(^1\)Summation convention assumed herein.
as,

$$(\nabla v)_{ij} = \frac{\partial v_i}{\partial x_j}, \quad (5.29)$$

$$(\nabla v)'_{ij} = \frac{\partial v'_i}{\partial x'_j} = \frac{\partial v_k}{\partial x_l} L_{ik} \frac{\partial v_k}{\partial x_l} L_{lj} = L_{ik} \frac{\partial v_k}{\partial x_l} L_{lj}, \quad (5.30)$$

$$(\nabla v)' = L(\nabla v)L^T. \quad (5.31)$$

Any portion of the $h$-box which is inside the solid region is reflected back into the fluid domain, and that reflected polygon is sliced in the same manner. The choice of plane through which to reflect seems trivial for the geometry illustrated in Figure 5.3; we choose the plane defined by the normal and a boundary edge point of the cut cell into which the $h$-box is pointing. When calculated from the reflected portion of an $h$-box, velocities and gradients are reflected using a Householder transformation,

$$H = I_3 - 2\hat{n}\hat{n}^T, \quad (5.32)$$

where $\hat{n}$ is the unit normal to the boundary through which the $h$-box was reflected and $I_3$ is a $3 \times 3$ identity matrix. For a segment reflected from inside a moving boundary, in a similar manner to Equation 4.7, solid wall boundary conditions are applied by adding twice the boundary velocity.

5.3.3 SLOPE RECONSTRUCTION AND LIMITING

To obtain a second-order solution, the gradient across each cell must be reconstructed. For fluid cells suitably far from cut cell boundaries, this is achieved using forward and backward gradients for each material parameter limited by the minmod limiter,

$$\begin{pmatrix} \partial_x \\ \partial_y \end{pmatrix}_{i,j} = \text{minmod} \begin{pmatrix} \Delta_{i+\frac{1}{2},j} \Delta_{i-\frac{1}{2},j} \\ \Delta_{i,j+\frac{1}{2}} \Delta_{i,j-\frac{1}{2}} \end{pmatrix}, \quad (5.33)$$

where $\Delta_{i+\frac{1}{2},j} = \frac{q_{i+1,j} - q_{i,j}}{\Delta x}$ is the gradient in the $x$ direction across the $(i + \frac{1}{2}, j)$ edge, and likewise for $y$. The minmod limiter is defined as,

$$\text{minmod}(a, b) = \begin{cases} 0 & ab < 0, \\ a & |a| \leq |b|, \\ b & \text{otherwise}. \end{cases} \quad (5.34)$$

---

2This derivation is from Riley et al. (2006).
This limiting scheme is appropriate for reconstructing gradients when none of the involved cells intersect the immersed boundary. For cut cells, edges which are fully inside the solid will not have a trivially computed gradient, and for cells neighbouring cut cells, the centroids will not necessarily be separated by $\Delta x$. For the affected cells, gradients are computed using a least squares method; the gradient is computed as the vector $(\partial_x, \partial_y)^T$ which minimises,

$$
\left\| \begin{pmatrix} (x_1 - x)^T \\ \vdots \\ (x_n - x)^T \end{pmatrix} \begin{pmatrix} \partial_x \\ \partial_y \end{pmatrix} - \begin{pmatrix} q_1 - q \\ \vdots \\ q_n - q \end{pmatrix} \right\|_2^2,
$$

with $x$ the centre of the cell, and $x_1,...,x_n$ the neighbouring cells. If the cell is a cut cell which only has two neighbours sharing an edge, then the diagonal cell is included as well.

To create a TVD scheme, these gradients must also be limited. To do this, the geometrical interpretation of minmod is necessary; in regular cells, the gradient is limited so as to avoid overshooting the value at adjacent cell centroids. This can be extended to cells without axis-aligned neighbours by requiring that,

$$
q + \left( \begin{array}{c} \partial_x \\ \partial_y \end{array} \right) \cdot (x_i - x) \leq q, \quad \forall i \leq n.
$$

Having obtained an estimate to $(\partial_x, \partial_y)^T$ using a least squares minimisation, there are multiple ways in which we might limit. One possibility is to multiply the entire gradient vector by a single scalar $\phi$ which enforces Equation 5.36; this has the advantage of being analogous with the normal 1D limiting approach, but a more subtle disadvantage. We may wish to consider reconstructing with a ghost version of the cut cell$^3$ included. However, for scalar variables ($\rho, p, &c$), this will lead to the conclusion that $\phi = 0$, and thus no reconstruction in the cut cells being performed.

This thought process reveals two things: firstly, that scalar variables should have limited gradients with no component in the direction normal to the surface; secondly, that a more clever limiter is necessary. As a simple alternative, for each cell used in the reconstruction, subtract only the portion of $(\partial_x, \partial_y)^T$ in the direction $(x_1 - x)^T$ which overshoots $q_i$. One disadvantage to this method is that the final result depends on the order in which the directions $x_1,...,x_n$ are limited.

$^3$That is to say, a ghost cell with position vector reflected through to the solid side of the boundary, and solid boundary conditions applied to its state vector.
5.3.4 PROBLEMS WITH THE \( h \)-BOX METHOD

Given how recently the simplified \( h \)-box method paper (Berger and Helzel, 2012) has been published, the literature is currently devoid of any other authors’ experiences using it. Despite some significant advantages, the method does come with its own set of problems. These break down into a few key categories:

- **Exactness of the flux cancellation property:** As described in this chapter, the \( h \)-box method critically relies on the cancellation between fluxes computed on a non-orthogonal grid between opposite pairs of edges in order to work around the small cell problem. Consider again Figure 5.3. All the \( h \)-boxes shown in the four figures are constructed using the normal to the boundary edge in cell \((i,j)\). Were the \((i,j - \frac{1}{2})\) flux calculated using the tangent to the boundary in cell \((i + 1,j - 1)\), it would not cancel with the flux through the \((i - \frac{1}{2},j)\) edge.

  So, the cancellation property only holds *exactly* for uniformly linear boundaries. This problem is reduced significantly by, at each edge, choosing the normal from the smaller of the two adjacent cut cells. This way the cancellation is maximised for the smaller cell (which is more critical to the stability of the scheme).

- **Handling of concave boundaries:** This relates to the previous problem, which is exacerbated significantly by sharply concave boundaries. The fix of making sure that the cancellation holds for the smaller adjacent cell can break down if a concave edge creates two adjacent small cells with drastically different normals. This problem must be fixed by merging small locally concave cells. Unfortunately, this means that a complete \( h \)-box implementation includes all the complexity of a cell merging code.

- **Amplification of numerical error:** The cancellation property aims for
\[
F_{i,j-\frac{1}{2}}^\xi - F_{i-\frac{1}{2},j}^\xi = \mathcal{O}(\alpha_{i,j}).
\]
Unfortunately this is suboptimal in terms of numerical stability. The cut cell area \(\alpha_{i,j}\) can be much smaller than the uncut cell area \(\alpha_{\text{max}}\); the numerical error in cut cells is amplified by a factor of \(\alpha_{i,j}/\alpha_{\text{max}}\). This may not be a problem for cells of relative size \(\sim 10^{-2}\), but it certainly becomes a prohibitive issue for sizes approaching machine epsilon\(^4\).

  However, again, this problem is mitigated; the value in small cells is never used directly, only as an average over an \( h \)-box which has area of order \(\alpha_{\text{max}}\). Even flagrantly non-physical values in tiny cells (such as negative density or pressure) will be “averaged out” by adjacent cell values. When interpreting results,

\(^4\)Machine epsilon \((\varepsilon_p)\) is the smallest number such that \((1 + \varepsilon_p)^* > 1\) where \(n^*\) denotes the floating point approximation to \(n\).
Results

however, one must take care not to place too much faith in values from extremely small cells\(^5\).

5.4 Results

In the previous chapter, problems with the ghost fluid method were highlighted. Here, we hope to show an improvement using the \(h\)-box method and a hybrid rotated-grid cell-merging approach for moving boundaries.

5.4.1 Static cylinder

One of the issues with the ghost fluid method was the creation of spurious oscillations at boundaries. Figure 5.4 shows a visualisation of the oscillations produced by both the ghost fluid and cut cell solid boundary methods soon after a shock wave is reflected by a cylinder\(^6\). The left side shows density and density schlieren snapshots using the \(h\)-box method, while the right shows the results from Figure 4.1. Figure 5.5 shows a line-out taken just downstream of the cylinder in the same simulation. The errors introduced by the ghost fluid method are visible even in the density field, which is not the case for the cut cell simulation. This represents a huge improvement.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5_4.png}
\caption{Top: density and bottom: density schlieren images just after the initial contact of the shock with the curved surface of the cylinder at \(t = 0.15\) s. For contrast, the images on the right show results produced with the ghost fluid method (from Figure 4.1), while the left were produced using the \(h\)-box method.}
\end{figure}

\(^5\)Note that in Equation 5.37, the error in each cut cell is weighted \(\propto \ell_b\), so large errors in small cells do not unduly affect the error calculation.

\(^6\)Full problem description in subsection 4.2.1, run at a resolution of 2000 \(\times\) 2000.
Cut cell methods

Line-out data for both methods are presented in Figure 5.6. Both methods converge to the same solution, although with different behaviour at low resolution. The difference in sharpness of the shock at low resolution is particularly apparent; this is due to the minmod limiter being more diffusive than van Leer limiter used for the ghost fluid simulation.

Figure 5.5: Density along the line $y = 0.395 \text{ m}$ at $t = 0.15 \text{ s}$ for the shock-cylinder impact simulation using top: the ghost fluid method, bottom: the $h$-box method.
Figure 5.6: Data taken along the line $y = 0.275$ m at $t = 0.4$ s for the shock–rigid cylinder impact case. Results are presented to contrast convergence between the ghost fluid and cut cell cases.
5.4.2 Steady-state aerofoil flow

For direct comparison with the ghost fluid method, the test case in subsection 4.2.2 is repeated using the $h$-box method using the same domain and number of cells. In the generated cut mesh, the smallest cut cell has area $\approx 4 \cdot 10^{-4}$ that of a normal grid cell. Significant improvement is seen when compared to the equivalent ghost fluid results; the spurious oscillation at the leading edge of the aerofoil are no longer present, and the recompression shock is captured with a smaller number of cells (although with a single cell undershoot).

![Figure 5.7: Pressure coefficient around a NACA 0012 aerofoil at zero angle of attack in Mach 0.8 flow, simulated to steady state using the $h$-box method. The line is produced by a high-resolution run with 200 cells along the aerofoil, while the points were generated with only 50 cells.](image)

5.4.3 Sloped channel flow

The convergence rate of any Riemann problem-based solver is best investigated using flows which do not contain discontinuities, as all such solvers are by necessity first-order in the vicinity of shocks due to Godunov’s theorem. The propagation of a density wave between two infinite planes at an angle of $20^\circ$ to the $x$ direction is modelled with initial conditions,

$$\rho = \begin{cases} 
2 - \cos^2 \left( \pi \frac{10\lambda - 1}{2} \right) & \frac{1}{10} \leq \lambda \leq \frac{3}{10}, \quad v_x = \frac{1}{2} \cos (20^\circ), \quad v_y = \frac{1}{2} \sin (20^\circ), \quad p = 1, \\
1 & \text{otherwise} 
\end{cases}$$

where $\lambda = x \cos (20^\circ) + y \sin (20^\circ)$ is the length along the plates. This produces a density wave which propagates at $|\mathbf{v}| = 0.5 \text{ m s}^{-1}$; as it is a simple advection problem,
the exact solution is given by $\rho(x, t) = \rho(x - vt, t = 0)$, and can be used for computation of the error at later time. The $L^n$ norm error at the boundary is given by,

$$L^n_{\partial V} = \left[ \sum_{i,j \in \partial V} |(\rho_{i,j} - \rho(x, t)) \ell_{i,j}| \right]^{\frac{1}{n}} \sum_{i,j \in \partial V} |\rho(x, t)\ell_{i,j}|,$$

(5.37)

where $\partial V$ is the set of cut cells, $\rho_{i,j}$ is the numerical solution at time $t$, $\rho(x, t)$ denotes the exact solution, and $\ell_{i,j}$ the length of the cut edge. Likewise, the error norm for the entire domain is,

$$L^n_V = \left[ \sum_{i,j \in V} |(\rho_{i,j} - \rho(x, t)) \alpha_{i,j}| \right]^{\frac{1}{n}} \sum_{i,j \in V} |\rho(x, t)\alpha_{i,j}|,$$

(5.38)

where $V$ is the set of all fluid cells, and $\alpha_{i,j}$ is the area of each cell. Figure 5.8 shows the error at $t = 0.5$ s for this test case using the original $h$-box method without gradient limiting. Near second-order convergence is maintained both at the boundary and in the bulk of the fluid; more importantly, the two regions converge to the exact solution at almost the same rate.

\[\begin{align*}
\text{Error} & \quad 0.1 \\
\Delta x & \quad 0.001 \quad 0.002 \quad 0.005 \quad 0.01 \quad 0.02 \quad 0.05
\end{align*}\]

Figure 5.8: Sloped channel advection test case using the $h$-box method without limiting. The best fit lines show near second-order convergence both at the boundary and in the entire fluid.
Moving cut cell methods

We now turn our attention to cut cell methods for moving immersed boundaries. As any algorithm for this must cope with a mesh which is deforming, a number of complications are introduced.

6.1 General method

Replacing the static solid obstruction with a time-varying object $V(t)$ leads to a number of complications. Most of the quantities established in section 5.1 become time-depandan, including the cut cell areas $\tilde{\alpha}_{i,j}(t)$ and edge lengths $\tilde{\ell}_{i,j}(t)$. We adopt a superscript to refer to time indices of these quantities. The conservation law the computational cell $(i,j)$ becomes,

$$
(\alpha U)_{i,j}^{n+1} = (\alpha U)_{i,j}^n - \Delta t \left[ (\ell F)_{i+\frac{1}{2},j}^{n+\frac{1}{2}} - (\ell F)_{i-\frac{1}{2},j}^{n+\frac{1}{2}} + (\ell F)_{i,j+\frac{1}{2}}^{n+\frac{1}{2}} - (\ell F)_{i,j-\frac{1}{2}}^{n+\frac{1}{2}} + (\ell F)_{i,j}^{n+\frac{1}{2}} \right],
$$

(6.1)

where the fractional time indices denote that the edge lengths should be averaged over the time interval $[t, t + \Delta t]$,

$$
\tilde{\ell}_{i-\frac{1}{2},j}^{n+\frac{1}{2}} = \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} \tilde{\ell}_{i-\frac{1}{2},j}(t) \, dt.
$$

(6.2)

These averaged edge lengths are referred to as “apertures” in the literature\(^1\). While conceptually simple, the accurate evaluation of these quantities is surprisingly challeng-

\(^1\)Or “space-time apertures” in Miller and Colella (2002).
General method

ing² for anything other than trivial geometries. For a linear boundary, these quantities can be obtained exactly. Barton et al. (2011) describes a procedure for extracting the cell apertures from a level set, which may prove useful as we are already equipped with efficient algorithms for manipulating level sets for use with the ghost fluid method.

For a moving boundary, the flux $F_{ij}$ contains a term corresponding to work done by the boundary segment on the fluid; $dW = F \cdot dx = \ell_{i,j}^{-1+\frac{1}{2}} p^* \hat{n} \cdot v dt$, where $p^*$ is the star state pressure obtained by solving a one-sided “piston” Riemann problem at the boundary. The total flux “through” the boundary is,

$$F_{i,j}^{n+\frac{1}{2}} = \begin{pmatrix} 0 \\ p^* \hat{n} \\ p^* \hat{n} \cdot v \end{pmatrix}. \quad (6.3)$$

Some information about the averaged boundary lengths can be deduced from requiring that the exact solution be recovered when the solid body moves with the same velocity as a uniform ambient flow. In such a test case, the solution in every fluid cell should be unchanging; that is,

$$U_{i,j}^{n+1} = U_{i,j}^n \forall i,j. \quad (6.4)$$

Applying this to the 2D Euler equations (Equation 2.1), including the moving boundary flux from Equation 6.3, and writing the velocity (of both the fluid and the solid boundary) as $v = (u, v)^T$ results in the four equations,

$$\Delta_t \tilde{\alpha}_{i,j}^n \rho = -\Delta t \left( \Delta_x \ell_{i,j}^{n+\frac{1}{2}} \rho u + \Delta_y \ell_{i,j}^{n+\frac{1}{2}} \rho v \right), \quad (6.5)$$

$$\Delta_t \tilde{\alpha}_{i,j}^n \rho u = -\Delta t \left( \Delta_x \ell_{i,j}^{n+\frac{1}{2}} (\rho u^2 + p) + \Delta_y \ell_{i,j}^{n+\frac{1}{2}} (\rho uv) - \ell_{i,j}^{n+\frac{1}{2}} p n_x \right), \quad (6.6)$$

$$\Delta_t \tilde{\alpha}_{i,j}^n \rho v = -\Delta t \left( \Delta_x \ell_{i,j}^{n+\frac{1}{2}} (\rho uv) + \Delta_y \ell_{i,j}^{n+\frac{1}{2}} (\rho v^2 + p) - \ell_{i,j}^{n+\frac{1}{2}} p n_y \right), \quad (6.7)$$

$$\Delta_t \tilde{\alpha}_{i,j}^n E = -\Delta t \left( \Delta_x \ell_{i,j}^{n+\frac{1}{2}} u (E + p) + \Delta_y \ell_{i,j}^{n+\frac{1}{2}} v (E + p) - \ell_{i,j}^{n+\frac{1}{2}} p n \cdot v \right), \quad (6.8)$$

where $\Delta_t \tilde{\alpha}_{i,j}^n = \tilde{\alpha}_{i,j}^{n+1} - \tilde{\alpha}_{i,j}^n$ is the volume change from $t_n$ to $t_{n+1}$; and $\Delta_x \ell_{i,j}^{n+\frac{1}{2}} = \ell_i^{n+\frac{1}{2}} - \ell_i^{n+\frac{1}{2}}$, and $\Delta_y \ell_{i,j}^{n+\frac{1}{2}} = \ell_j^{n+\frac{1}{2}} - \ell_j^{n+\frac{1}{2}}$ are the differences between opposite edge lengths in $x$ and $y$ respectively. The star state boundary pressure $p^* = p$ for the case of uniform flow. These three consistency equations are not independent, with only three

Indeed, a significant portion of Falcovitz et al. (1997) is devoted to the topic
Moving cut cell methods

degrees of freedom; solving for $\Delta t \tilde{\alpha}_{i,j}^n$, $\Delta x^{\tilde{\alpha} n + \frac{1}{2}}_{i,j}$, and $\Delta y^{\tilde{\alpha} n + \frac{1}{2}}_{i,j}$ yields,

\[
\Delta t \tilde{\alpha}^n_{i,j} = \frac{\tilde{\alpha}^n_{i,j} - \tilde{\alpha}^{n+1}_{i,j}}{v \cdot \hat{n}},
\]

\[
\Delta x^{\tilde{\alpha} n + \frac{1}{2}}_{i,j} = \frac{\tilde{\alpha}^n_{i,j} - \tilde{\alpha}^{n+1}_{i,j}}{v \cdot \hat{n}} - n_x,
\]

\[
\Delta y^{\tilde{\alpha} n + \frac{1}{2}}_{i,j} = \frac{\tilde{\alpha}^n_{i,j} - \tilde{\alpha}^{n+1}_{i,j}}{v \cdot \hat{n}} - n_y.
\]

These consistency conditions deserve significant consideration. In order to preserve the exact solution in the case of a solid body moving at the same velocity as a uniform ambient flow, these must be exactly satisfied by the time-averaged boundary lengths. Unfortunately, the procedure for discretising the geometry described in section 5.2 leads to only a piecewise linear representation. This means that for any non-linear boundary, the area of the solid region enclosed will fluctuate as it traverses the domain; therefore, for a complex piecewise-linear boundary, we cannot simultaneously have both conservation and consistency.

As the areas of cells change between time steps, Equation 2.31 will no longer be conservative. Therefore, the averaging procedure is modified to include area-weighting,

\[
(\alpha q_l)^{n+1}_i = (\alpha q_l)^n_i + (\alpha q_l)^{(2)}_i.
\]

6.2 Piston flux applied naïvely to the $h$-box method

Consider a one dimensional piston problem consisting of a uniform initial state $q = (\rho, 0, p)^T$ with a piston represented by a moving cut boundary through the left cell (illustrated in Figure 6.1). At time $t_n$ the cut cell has length $\alpha^n$, reducing to $\alpha^{n+1}$ at the next step. Taking the density component of Equation 6.1 yields a change in

Figure 6.1: Simulation set-up corresponding to a cut cell being compressed by a solid boundary moving across it. The region to the left of the cut is outside of the fluid domain, but is assigned flowfield values using standard moving wall boundary conditions.
density of,

$$\rho^{n+1} - \rho^n = \frac{\alpha^n - \alpha^{n+1}}{\alpha^{n+1}} \rho - \frac{\Delta t}{\alpha^{n+1}} F(q_L, q_R), \quad (6.13)$$

where \( F \) denotes the solution to the Riemann problem with given left and right states.

Using the \( h \)-box method, the left state is found by averaging \( q(x) \) over the cut cell. Using the boundary conditions for a moving solid wall gives a virtual state with velocity \( 2v \) in the solid region of the cut cell, so averaged over the left cell, \( v_L = (1 - \alpha^n) 2v \).

Putting this together with an expression for the change in length of the small cell, \( \alpha^n - \alpha^{n+1} = v \Delta t \), gives a total expression for the change in density in the cut cell,

$$\rho^{n+1} - \rho^n = \frac{\Delta t}{\alpha^{n+1}} \left[ \rho v - F \left\{ \begin{pmatrix} \rho \\ (1 - \alpha^n) 2v \\ \rho \end{pmatrix}, \begin{pmatrix} \rho \\ 0 \\ p \end{pmatrix} \right\} \right]. \quad (6.14)$$

Figure 6.2 shows the evaluation of this equation with initial conditions \( \rho = p = 1 \), \( v = 0.1 \), \( \Delta t = 0.5 \) for varying end sizes of the cut cell \( \alpha^{n+1} \). Note that the density change tends to \(-\infty\) as \( \alpha^{n+1} \to 0 \), meaning that the method is not stable for small cells. So, unfortunately, this extension of the \( h \)-box method to moving boundaries is a dead end.3

![Figure 6.2](image-url)
6.3 CELL MERGING & THE ROTATED GRID METHOD

This last result is profoundly discouraging for a moving cut cell method which attempts to retain small cells. Perhaps inspiration can be drawn from one of the precursors to Berger and Helzel (2012). In Helzel et al. (2005), Christiane Helzel investigated the use of the rotated grid method to improve the accuracy of strongly directional flows on Cartesian grids—not just for embedded boundaries. This raises the possibility of using a different cut cell method to work around the small cell problem, while using the rotated grid method (still aligned with the boundary) to improve accuracy.

Cell merging is a simpler method to implement in 2D than the \( h \)-box method, although its use with moving boundaries introduces a host of complications. For example, edges not in the fluid domain at time \( t_n \) may appear at \( t_{n+1} \), and \textit{vice versa}. The principle of the method is to merge cells with area smaller than a minimum selected by the user (\( \alpha_{\text{min}} \)) with adjacent cells (which be either cut or whole) to create larger cells which do not reduce the stable time step by too large a factor. For the purposes of this work, we make the common choice of \( \alpha_{\text{min}} = 0.5 \); of course, this does depress the stable timestep by a factor of two.

For reasonably smooth non-concave 2D geometries, it generally suffices to pair cells below the area threshold with adjacent cells in the cardinal direction nearest to the direction of the normal. This results in a logical domain with cell sizes in the range \( 0.5\alpha_{\text{uncut}} \) to \( 1.5\alpha_{\text{uncut}} \). The modification to the numerical scheme to accommodate merged cells is simple; the logical requirement is that Equation 6.1 be modified to include the flux through all exposed edges of the merged pair of cells. This can easily

\[ \text{In fact, there is another problem with this approach; in Equation 6.1, the flux is divided by } \alpha^{n+1}, \text{ while the } h \text{-box method constructed on the grid at time } t_n \text{ introduces a factor of } \alpha^n \text{ to cancel. For a moving boundary, these areas may differ by a large factor.} \]
be achieved by performing the flux update for each cell, disregarding merges, and performing an area weighted average of the conserved variables for each cell \((i,j)\) merged with cell \((i',j')\),

\[
(\alpha_{i,j} + \alpha_{i',j'})q_{i,j} \leftarrow \frac{1}{2} \left( \alpha_{i,j} q_{i,j} + \alpha_{i',j'} q_{i',j'} \right),
\]

\[
q_{i',j'} \leftarrow q_{i,j}.
\]

(6.15)

As the edge joining cells \((i,j)\) and \((i',j')\) has an outward facing normal which is opposite with respect to each cell, the flux through the “internal” edge cancels exactly. Note that no explicit procedure is necessary for unmerging a pair of cells; as they have both been assigned the same fluid values, it suffices just to remove the pairing.

Care must be taken when choosing which cells to include in the linear reconstruction. When reconstructing in a merged cell, we use both cells adjacent to the merged edge. This leads to a maximum of five adjacent cells for the minimisation routine.

### 6.4 Results

![Convergence plot](image)

Figure 6.4: Convergence plot for the linear advection test case using cell merging to eliminate cut cells with \(\alpha < 0.5\). A linear reconstruction in cut cells is attempted using a least squares limiter, but the gradients obtained are not limited.

On the way to developing a moving cut cell method involving cell merging, it is also useful to consider the convergence properties of a stationary cell merging algorithm. Figure 6.4 shows the results of these tests. Again, the \(L^2\) norm for both the entire fluid domain and at just the boundary are shown. The whole domain shows a similar rate of
convergence; however, below $\Delta x \approx 0.002$, the convergence rate at the boundary drops dramatically.

The cause of this error is not obvious. This shape of convergence plot is common when encountering issues of numerical precision. If a simulation has hit some kind of limiting factor due to the accumulation of numerical error, the convergence plot will typically plateau\(^4\). We might imagine that arbitrarily small edge lengths (or similar) are causing problems beyond a certain resolution, though it is hard to imagine what problems could arise in a cell merging code that would not be worse with the $h$-box method.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6_5.png}
\caption{Convergence plot for the same simulation as Figure 6.4, but with a moving boundary, and limited reconstructed gradients.}
\end{figure}

To adjust the test to moving boundary problems, we move the boundary with velocity $v_b = 0.5$ in the $x$ direction, and adjust the fluid velocity to $v_x = \frac{1}{2} \cos(20^\circ) + v_b$ to recover an advection problem. Figure 6.5 shows a convergence plot for this moving boundary problem. This time, a limiter proved necessary, so gradients in all cells were limited using the minmod limiter. The dashed line is proportional to a second-order rate of convergence. The use of a limiter reduces the rate of convergence to $O(\Delta x^{1.29})$, but importantly, the boundary cells converge at the same rate as the bulk of the fluid until the same convergence glitch occurs as in the stationary cell merging case.

\(^4\text{Consider the most simple limit: we would not expect the error in any cell to be less than numerical epsilon, aside from particularly serendipitous cases.}\)
Figure 6.6: The error in density for a $800 \times 400$ simulation along a moving ramp; $+$ denotes the numerical solution, the solid line the exact solution, and the relative error is shown underneath. This test case uses a limiter, so there is a slight rounding-off of the peak of the cosine bell. However, the dominant error source is a phase error.
Chapter 7

Conclusions

The previous two chapters present a number of new results relating to moving boundary methods. These break down into uses of existing static boundary methods, and a new moving boundary method.

7.1 Stationary cut cell methods

This work independently validates the results of Berger and Helzel (2012), which proposes the use of the rotated grid method from Helzel et al. (2005) to avoid the small cell problem associated with cut cell algorithms. Figure 5.8 confirms that the $h$-box method provides a second-order solution at immersed boundaries. To our knowledge, this is the only scheme to provide the same order of convergence in cut cells and normal cells.

However, we also report a number of significant complications and drawbacks of the method. The code developed during this work has uncovered a number of issues which need further work before it would be robust enough for production use. Perhaps the most fundamental issue is mesh generation; this work has been restricted to simple geometries with analytic expressions. For general use, we will need to be able to input arbitrary geometries, and avoid any incorrectly cut edges; this may require use of an arbitrary precision arithmetic library. One possible path is to use a similar technique to Barton et al. (2011) for calculating edge apertures and cell areas from a level set.

For some applications, spurious oscillations introduced by the ghost fluid method
Moving cut cell methods

are a significant problem. The example provided for this work is critical flow around an aerofoil; similar problems have been observed in unsteady simulations of turbomachinery. Oscillations produced by moving turbines lead to very noisy output for torque calculations. The better representation of the boundary provided by the $h$-box method reduces these oscillations to an imperceptible level.

7.2 Moving cut cell methods

The results presented in this work show the possibility of a moving cut cell method with super-linear convergence at the boundary. Further investigation is certainly required to establish why convergence slows above a certain mesh size, but for lower resolution, the convergence rate at the boundary of the domain is measured to be almost identical to the bulk of the flow.

We intend to implement this method in a larger multimaterial hydrocode using the methods described in Barton et al. (2011) to compute the cut cell mesh from an evolving level set.

7.3 Further work

• Implement in AMR code: rounding-off of sharp edges when generating cut cell meshes can be reduced using adaptive mesh refinement. A first step to facilitate further investigation would be to implement the algorithms described in Chapter 5 in the Laboratory for Scientific Computing’s AMR framework.

• More complex boundaries: the moving boundary algorithm developed in this dissertation is heavily optimised for linear boundaries. Extending it to complex geometries should be relatively straightforward, as the only issue is accurate calculation of cell face apertures.

• LP limiter: a preprint by May and Berger describes a new method of limiting on irregular grids which. Rather than estimating the gradient and then limiting, the gradient calculated by maximising $(\partial_\xi, \partial_\eta)$ subject to the TVD constraint (Equation 5.36). This reduces any directional biasing of the reconstruction, which is one possible cause for the convergence glitch.

• Non-conservative flux: Miller and Colella (2002) constructs a cut cell method which uses a non-conservative flux through boundary edges combined with a conservation fix-up. This may provide a way to avoid the problem described in section 6.2, and allow the construction of a moving cut cell method which retains small cells.


X. Y. Hu, N. A. Adams, and G. Iaccarino. On the HLLC Riemann solver for interface...


REFERENCES


