Master's thesis

2019

Frederik Kristoffersen

Master's thesis

NTNU
Norwegian University of
Science and Technology
Faculty of Engineering
Department of Energy and Process Engineering

NTNU
Norwegian University of
Science and Technology

Frederik Kristoffersen

Adaptive Mesh Refinement for the Euler Equations of Gas Dynamics

June 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Adaptive Mesh Refinement for the Euler Equations of Gas Dynamics

Frederik Kristoffersen

Mechanical Engineering
Submission date: June 2019
Supervisor: Bernhard Müller

Norwegian University of Science and Technology
Department of Energy and Process Engineering

Abstract

A finite volume solver that utilizes adaptive mesh refinement (AMR) on unstructured Cartesian grids has been developed for the two-dimensional Euler equations of gas dynamics. The solver uses the explicit Euler method and the Rusanov method for time and flux discretization, respectively. The rectangular cells can be refined by quadrisection through their centers. This preserves their aspect ratios and doubles the spatial resolution locally. Four cells that were created from the same cell refinement can be merged by reversing the refinement process. The criterion for refinement and merging is based on the absolute differences of the density and velocity components in neighboring cells. For triggering adaptation, i.e., for deciding when to perform AMR, a new criterion is proposed. It is based on accumulating the absolute rate of change of mass relative to its initial value.

The development of a regular oblique shock reflection from a plane wall is simulated, starting from an initial condition corresponding to a Riemann problem. For comparison, the Euler equations of gas dynamics are also solved by a standard finite volume solver on a structured Cartesian grid. Using AMR, the number of cells can be reduced by up to 95%, i.e. a factor 20, to achieve the same error as the standard finite volume solver. Even though mesh adaptation impairs convergence to steady state and there is some overhead related to the data structure, the AMR solver takes only 36% of the computing time needed by the standard solver, in the most beneficial cases. The potential of the AMR solver for unsteady flow is demonstrated for the simulation of a 2D Riemann problem.

Acknowledgement

I want to thank my academic supervisor, Professor Bernhard Müller. He has been actively involved in the project from its start, and dedicated much time to provide greatly appreciated counsel, encouragement, feedback, ideas and critique. He also initiated my interest in the field of computational fluid dynamics. I am truly grateful for all of this. Thank you, Bernd!

I also give a big thanks to my girlfriend and my family, for their support, and for always believing in me.

I thank all developers and providers of free open-source software. Because of them, anybody with the know-how and a functioning computer could produce the results and figures presented in this report.

Contents

List of Figures

List of Tables

Description of the Cover Figure

The figure on the cover of this document shows a plotted numerical solution of the horizontal velocity component u. The color red represents the highest values, and blue represents the lowest values. There are two narrow areas where the color changes rapidly, which indicates that there are discontinuities in the solution. The black lines are the mesh faces that divide the domain into square-shaped cells. We can see that the cells around the discontinuities are much smaller than elsewhere. These cells have been refined when the mesh was adapted to the solution.

Nomenclature

Symbols	
α , β	Shock angles, cf. Figures 2, 4 and 5.
γ	Ratio of specific heats, cf. equation (10).
δ_r , δ_m	Refine and merge thresholds, cf. equations (136) and (137).
Δ	Difference between two values.
ε_ρ	Numerical error, cf. equation (131).
η_r , η_m	Refine and merge tolerances, cf. equations (136) and (137).
θ	Shock angle, cf. Figure 3 and Table 1.
ρ	Density of mass.
$\sum$	Sum.
τ	CPU time, runtime. The time it took to run a simulation.
φ	Velocity deflection angle, cf. Figure 3 and Table 1.
Ω	Arbitrary control volume.
a , b	Characteristic velocities in the x- and y-directions respectively, cf. equation (114).
A	Area.
c	Speed of sound, cf. equation (11).
c_p , c_v	Specific heats at constant pressure and volume, respectively.
C_{max}	Maximum occurring Courant number, cf. equation (111).
E	Specific total energy.
$\mathbf{F}$, $\mathbf{G}$	Flux vectors in the x- and y-directions, respectively, cf. equations (24) and (113).
i	A cell. Subscript denotes a value in a cell, cf. Figure 8.
j, k, l, m	Face number in data structure, cf. Figure 15.
k	A face. Subscript denotes a value at a face, cf. Figure 8.
L	Refinement level, cf. Figure 1.
$_L$	Subscript denoting adjacent cell, cf. Figure 8.
m , $\tilde{m}$	Merge flag and prevent merge flag, cf. equations (141), (142).
M	Mach number, cf. equation (12).
n	Time level, cf. subsection 4.2. Superscript denotes variable at given time level.
N_i	Number of cells in the mesh.

Symbols, continued	
p	Thermodynamic pressure.
q	Grading degree, cf. Figure 1.
r	Refine flag, cf. equation (139).
R	Specific gas constant of the ideal gas law.
$_R$	Subscript denoting adjacent cell, cf. Figure 8.
t	Time. Not CPU time.
T	Temperature.
u	Velocity component parallell with the x-axis.
$\mathbf{U}$	Vector of conserved variables, cf. equation (23).
v	Velocity component parallell with the y-axis.
$\mathbf{V}$	Vector of primitive variables, cf. equation (25).
V	Volume.
w	Modulus of velocity, cf. Figure 3 and Table 1.
x , y	Spatial coordinates.

Acronyms and Abbreviations	
aka	Also known as
AMR	Adaptive mesh refinement
BC	Boundary condition(s)
cf.	conferatur, "as in"
CFD	Computational fluid dynamics
CPU	Central processing unit
E.g.	Exempli gratia, "for example"
etc.	et cetera, "and so on"
FVM	Finite volume method
IC	Initial condition(s)
I.e.	Id est, "that is" or "in other words"

1 Introduction

The scope of this report is similar to my previous work, *Adaptive Mesh Refinement in Conservation laws* [1]. I therefore reuse parts of the motivation and background from said work. Although I have made several modifications in notation and phrasing, much of subsection 1.1 is taken from [1]. Throughout this report, several of the figures are taken from [1] and adapted slightly.

1.1 Background and Motivation

In computational fluid dynamics (CFD), the demand for high mesh resolution is not equal throughout the computational domain. High gradients or discontinuities in flow variables require higher resolution, i.e. smaller and more densely packed cells. It is possible to do this adaptation before simulating, if a priori knowledge, or justified assumptions are available for the flow. In many flows the regions that require high resolution are moving. Examples are vortex shedding, and moving shocks. It would be possible to resolve the vortices or shock by refining the mesh in the entire region the features traverse before they dissipate. Another approach, which can lower the computational cost, is to detect resolution demanding features while simulating, and refine or coarsen accordingly [2–6]. This approach is known as *Adaptive Mesh Refinement*, or AMR for short.

An adaptation strategy describes how the AMR is executed on an algorithmic level. It should answer the questions: *when*, *where* and *how* to adapt?[7] The choice of adaptation strategy is closely linked with the choice of mesh, and affects how the mesh should be generated. The AMR solver developed for this project, uses an unstructured Cartesian mesh. Marsha J. Berger has contributed to the field of AMR using unstructured Cartesian meshes in cooperation with Michael J. Aftosmis and John E. Melton [8]. This work has laid the basis for locally refining AMR softwares such as NASA's Cart3D project [9].

One method for refining a Cartesian mesh locally, is to quadrisect the cells through their centers [4, 10]. This allows the refinement state of the cells to be classified by levels [11, 12]. These refinement levels L correspond to cell sizes, as shown in Figure 1. The largest permitted cell size corresponds to level $L = 0$, and gives the lowest possible resolution. The smallest allowed cell size corresponds to $L = L_{max}$, which gives the highest possible resolution. Each time L increases by one, the spatial resolution is locally doubled in all dimensions. This means that when a 2D cell is refined, it is split into four geometrically similar cells [4, 10], as can be seen in Figure 1. In other words, all the cells have equal aspect ratio, regardless of their refinement levels. For three-dimensional meshes, refined cells are split into eight cells. Only 2D cases are considered in this project.

An advantage of using this refinement approach is flexibility. Any cell can be split into four, as

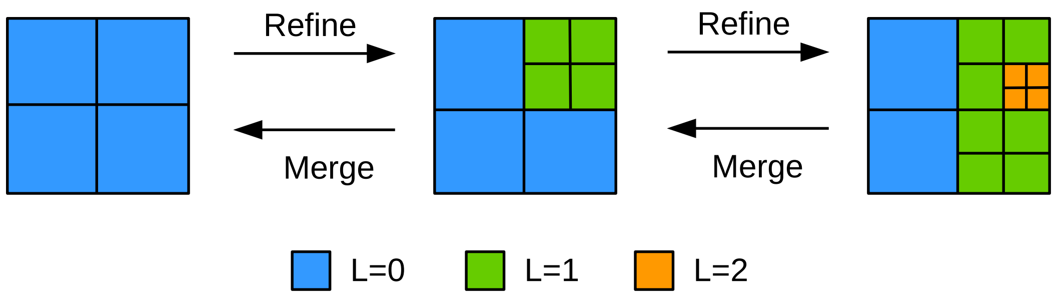


Figure 1: Principle sketch of local grid refinement and merging using quadrisection. L is the cell refinement level. The grading degree is $q = 1$, which means that between a cell at $L = 0$ and a cell at $L = 2$, there must be at least one cell at $L = 1$. This figure is taken from [1] and adapted slightly.

can be seen in Figure 1. Conversely, four cells that were created by the same cell refinement can be merged. However, it is advantageous to limit this freedom with a constraint called *grading* [11]. This constraint sets the lower limit, denoted by q, for how many cells must come between two level changes. With a grading degree of $q = 0$, there is no limit to how close two level changes can be. Essentially, this means that the refinement level can change multiple times from one cell to another, i.e. a very small cell can be adjacent to a big cell. This can negatively affect accuracy, and it can make the local refining and coarsening algorithms more intricate. With a grading degree of $q = 1$, like in Figure 1, there needs to be at least one cell between level changes. This means that the resolution level can change for every traversed cell, but only *one* level at a time.

Another approach of mesh refinement is known as structured AMR, or S-AMR for short [2, 4, 8]. This method involves organizing the computational domain into patches. All the cells that belong to the same patch have the same resolution. This makes the detection and refinement procedures much simpler [2], which is a an advantage. It also makes it much easier to implement methods with a higher order of accuracy in space, due to the fact that each patch is a structured sub-mesh. However, many cells will get a higher resolution than they require, unless the patches are very small. If the patches are too small, then the maximum cell size is limited, and the inter-patch communication becomes costly. The work of Marsha J. Berger must be mentioned here as well. She has contributed to the field of S-AMR, collaborating with Randall J. LeVeque. Their work in this field is utilized in the AMRCLAW software package [2], which is part the CLAWPACK package [13].

A third alternative is to refine locally using a tree structure [14], instead of an unstructured mesh. For a 2D solver it could be a quadtree structure. This approach allows for an intuitive data structure, if combined with the quadrisection method discussed above. However, traversing the leaf nodes in a tree data structure is costly, unless an additional mapping of the leaf nodes is added. Another option is to use hash tables [3, 14]. Neither S-AMR, tree structures or hash tables are used in this project, but they are listed as alternatives.

There are multiple ways to detect where the mesh needs refinement, and where it can be coarsened. One way is to compare the entire solution from the latest time step, with a solution calculated on a coarsened mesh [2]. If the error of a cell, or patch of cells, is above a given threshold then the cell or patch is marked for refining. Another way is to evaluate the gradients in the solution, in pairs of neighbor cells. Then the cell or patch could be flagged for refinement or merging based on gradient thresholds. In this project, the solver checks differences between flow variables in neighboring cells. Regardless of how the detection is done, it will consume CPU time. Therefore it is common not to detect at every time step, but to have a criterion for *when* to adapt. A simple criterion is to specify a number of time steps between each detection routine [2], or a time interval. The AMR-solver in this project decides when to adapt, by accumulating an approximated mass redistribution rate. This will be elaborated in subsection 5.2.

1.2 Problem Description

I quote the objectives given by my academic supervisor, Professor Bernhard Müller:

> *" The goal of the project is to devise, implement and test adaptive mesh refinement and coarsening for the Euler equations of gas dynamics. The research is most relevant for the development of accurate and efficient numerical methods in CFD.*
>
> *The following tasks are to be considered:*
>
> 1. *to get a basic understanding of adaptive mesh refinement (AMR), solution based error criteria, and strategies to refine and coarsen the mesh for the Euler equations of gas dynamics,*
> 2. *to devise, implement and test adaptive mesh refinement for the Euler equations of gas dynamics,*
> 3. *to investigate the performance of AMR for the Euler equations of gas dynamics with different error criteria,*
> 4. *to compare the accuracy and CPU time of numerical solutions of the Euler equations of gas dynamics with and without adaptive mesh refinement. "*

— Bernhard Müller, Department of Energy and Process Engineering, NTNU
January 15, 2019

1.3 Outline of this Report

This thesis will continue with the following sections: In section 2, the 2D Euler equations of gas dynamics are outlined, and an equation of state is given. In section 3 two test cases are introduced and their initial and boundary conditions are stated. The first one is a regular oblique shock reflection over a plane wall, for which an analytical solution is derived. The second case is a 2D Riemann problem. The explicit finite volume method (FVM) for the discretization of the 2D Euler equations is given in section 4. The present AMR approach based on unstructured Cartesian grids is outlined in section 5. Results for the two test cases are presented and discussed in section 6. The accuracies and efficiencies of the AMR solver and a standard solver are compared in subsection 6.2. In section 7 conclusions are drawn and section 8 outlines the future outlook for the project.

2 The Euler Equations of Gas Dynamics

Many gas dynamics problems include discontinuities and areas with constant flow variables. Thus, they can achieve large efficiency gains from the AMR approach. The AMR solver presented in this report specializes in solving inviscid gas flow problems, governed by the Euler equations of gas dynamics. These equations model the conservation of mass, momentum and total energy. The continuity equation,

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0 \quad , \tag{1}$$

models the conservation of mass. The density, ρ, is the conserved variable, u is the velocity component in the x-direction, and v is the velocity component in the y-direction. Secondly, the horizontal momentum-, or x-momentum equation,

$$\frac{\partial(\rho u)}{\partial t} + \frac{\partial}{\partial x}\left[\rho u^2 + p\right] + \frac{\partial}{\partial y}\left[\rho v u\right] = 0 \quad , \tag{2}$$

models the conservation of momentum in the x-direction. ρu is the x-momentum density and p is the pressure. Thirdly, the vertical momentum-, or y-momentum equation,

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial}{\partial x}\left[\rho u v\right] + \frac{\partial}{\partial y}\left[\rho v^2 + p\right] = 0 \quad , \tag{3}$$

models the conservation of momentum in the y-direction. ρv is the y-momentum density. Fourth and final is the total energy equation,

$$\frac{\partial(\rho E)}{\partial t} + \frac{\partial}{\partial x}\left[u(p + \rho E)\right] + \frac{\partial}{\partial y}\left[v(p + \rho E)\right] = 0 \quad , \tag{4}$$

which models the conservation of total energy. ρE is the total energy density, and

$$E = e + \frac{1}{2}\left(u^2 + v^2\right) \tag{5}$$

is the specific total energy, where e is the specific internal energy.

In addition to these four flow equations, two equations of state are required to give two relations between density ρ, pressure p, temperature T and internal energy e. For this purpose, ideal gas is assumed. The equations of state for perfect gas read [15]:

$$p = \rho R T \quad , \tag{6}$$

$$e = c_v T \quad , \tag{7}$$

where R is the specific gas constant

$$R = c_p - c_v \quad . \tag{8}$$

c_p and c_v are the specific heats at constant pressure and volume, respectively. The specific heats are assumed to be constant. The equations of state yield an explicit expression for the pressure,

$$p = (\gamma - 1)\left[\rho E - \frac{1}{2}\rho\left(u^2 + v^2\right)\right] \quad , \tag{9}$$

which we will utilize to express the pressure as a function of ρ, ρu, ρv and ρE.

$$\gamma = \frac{c_p}{c_v} \tag{10}$$

is the ratio between the specific heats, c_p and c_v. Since c_p and c_v are assumed to be constant, γ is constant. We shall use $\gamma = 1.4$ for air at standard conditions. The speed of sound for perfect gas is given by

$$c = \sqrt{\gamma \frac{p}{\rho}} \quad , \tag{11}$$

which will be needed for the numerical method, and to compute the Mach number.

$$M = \frac{w}{c} \quad , \tag{12}$$

where

$$w = |\mathbf{w}| = \sqrt{u^2 + v^2} \tag{13}$$

is the modulus of the velocity vector $\mathbf{w} = [\, u \,, \;\; v \,]^T$.

To make the results easier to compare with other results, the governing equations should be non-dimensionalized. The individual variables are non-dimensionalized as follows:

$$
\begin{array}{llll}
x^* = \frac{x}{x_{max}} & , & y^* = \frac{y}{x_{max}} & , \\
u^* = \frac{u}{u_\infty} & , & v^* = \frac{v}{u_\infty} & , \\
\rho^* = \frac{\rho}{\rho_\infty} & , & p^* = \frac{p}{\rho_\infty u_\infty^2} & , \\
t^* = t^* \frac{u_\infty}{L} & , & c^* = \frac{c}{u_\infty} & , \\
E^* = \frac{E}{u_\infty^2} & , & &
\end{array}
\tag{14}
$$

where the superscript $*$ denotes dimensionless variables, and the length scale x_{max} is the length of the domain. The velocity scale u_∞ and the density scale ρ_∞ are specific to the test cases, discussed in subsections 3.1 and 3.2. The variables from (14) can be substituted into the governing equations, starting with the continuity equation (1):

$$
\frac{\rho_\infty \partial \rho^*}{\frac{L}{u_\infty}\partial t^*} + \frac{\rho_\infty u_\infty \partial(\rho^* u^*)}{L \partial x^*} + \frac{\rho_\infty u_\infty \partial(\rho^* v^*)}{L \partial y^*} = 0 \quad . \tag{15}
$$

$\frac{\rho_\infty u_\infty}{L}$ is a common factor and can be removed, yielding the non-dimensionalized continuity equation,

$$
\frac{\partial \rho^*}{\partial t^*} + \frac{\partial(\rho^* u^*)}{\partial x^*} + \frac{\partial(\rho^* v^*)}{\partial y^*} = 0 \quad . \tag{16}
$$

The same procedure is applied to the x-momentum equation. I.e. the variables (14) are substituted into (2), and the common factor $\frac{\rho_\infty u_\infty^2}{L}$ is removed. This yields the non-dimensional x-momentum equation,

$$
\frac{\partial(\rho^* u^*)}{\partial t^*} + \frac{\partial}{\partial x^*}\left[\rho^* u^{*2} + p^*\right] + \frac{\partial}{\partial y^*}\left[\rho^* v^* u^*\right] = 0 \quad . \tag{17}
$$

The exact same procedure is applied again to the y-momentum equation (3), to give the non-dimensional y-momentum equation,

$$
\frac{\partial(\rho^* v^*)}{\partial t^*} + \frac{\partial}{\partial x^*}\left[\rho u^* v^*\right] + \frac{\partial}{\partial y^*}\left[\rho^* v^{*2} + p^*\right] = 0 \quad , \tag{18}
$$

and to the total energy equation (4), yielding the non-dimensional total energy equation,

$$
\frac{\partial(\rho^* E^*)}{\partial t^*} + \frac{\partial}{\partial x^*}\left[u^*(p^* + \rho^* E^*)\right] + \frac{\partial}{\partial y^*}\left[v^*(p^* + \rho^* E^*)\right] = 0 \quad . \tag{19}
$$

The equations of state are also non-dimensionalized in this way, giving the non-dimensional pressure formula,

$$p^* = (\gamma - 1)\left[\rho^* E^* - \frac{1}{2}\rho^*\left(u^{*2} + v^{*2}\right)\right] \quad , \tag{20}$$

and the non-dimensional speed of sound,

$$c^* = \sqrt{\gamma \frac{p^*}{\rho^*}} \quad . \tag{21}$$

All the non-dimensionalized equations assume the exact same form as the dimensional equations, except for the superscripts $*$. Therefore, and to simplify the notation, the superscripts $*$ are omitted in the rest of this report. Thus, x, y, t, ρ, u, v, p, c, etc. denote non-dimensional variables.

To further simplify the notation, the flow equations (16), (17), (18) and (19) are combined in a vector equation:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial x} + \frac{\partial \mathbf{G}(\mathbf{U})}{\partial y} = 0 \quad , \tag{22}$$

where

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix} \tag{23}$$

is the vector of conservative variables and

$$\mathbf{F}(\mathbf{U}) = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ u(p + \rho E) \end{bmatrix} \quad \text{and} \quad \mathbf{G}(\mathbf{U}) = \begin{bmatrix} \rho v \\ \rho v u \\ \rho v^2 + p \\ v(p + \rho E) \end{bmatrix} \tag{24}$$

are the flux vectors in the x- and y-directions, respectively. The density ρ, the velocity components u and v, and the pressure p will be referred to as *primitive variables*. They can be assembled into the vector of primitive variables:

$$\mathbf{V} = \begin{bmatrix} \rho \\ u \\ v \\ p \end{bmatrix} \tag{25}$$

The primitive variables (25) and the Mach number (12) will be in focus when solutions are presented, and when solution accuracies are measured.

3 Test Cases

Two test cases were considered: regular oblique shock reflection [10, 16], for which the analytical solution is outlined in subsection 3.1, and a 2D Riemann problem [17], which is defined in subsection 3.2.

3.1 Regular Oblique Shock Reflection

In this subsection a regular oblique shock reflection from a plane solid wall will be introduced. First the flow will be described, to give an intuitive overview. Then, the oblique shock relations will be used to derive the analytical solution for the flow.

We have a supersonic gas flow over a solid wall. As illustrated in Figure 2, an incoming oblique shock, s_α, enters through the left boundary at $y = y_\alpha$. The shock s_α strikes the wall at the lower boundary at x_R, causing a regular shock reflection. The reflected shock, s_β, exits the domain through the right boundary. The shocks then form the angles α and β with the wall, which are also shown in Figure 2. The domain can then be divided into three regions, using the shock characteristics $s_\alpha(x)$ and $s_\beta(x)$. Region 1 is on the lower left side (upstream) of the incoming shock s_α:

$$0 \leq x_1 \leq x_R \quad \text{and} \quad 0 \leq y_1 \leq s_\alpha(x_1) \tag{26}$$

Region 2 is between s_α and the reflected shock s_β,

$$0 \leq x_2 \leq x_{max} \quad \text{and} \quad s_\alpha(x_2) < y_2 \geq s_\beta(x_2) \quad \text{and} \quad y_2 \leq y_{max} \quad , \tag{27}$$

and region 3 is on the lower right side (downstream) of s_β:

$$x_R < x_3 \leq x_{max} \quad \text{and} \quad 0 \leq y_3 < s_\beta(x_3) \tag{28}$$

x_{max} and y_{max} mark the right and upper boundaries, respectively. Subscripts 1, 2 and 3 denote variables in regions 1, 2 and 3, respectively. The shock characteristics and regions are also shown in Figure 2. Inside these regions, all the flow variables will be constant. Thus, the primitive variables (25) can be expressed as

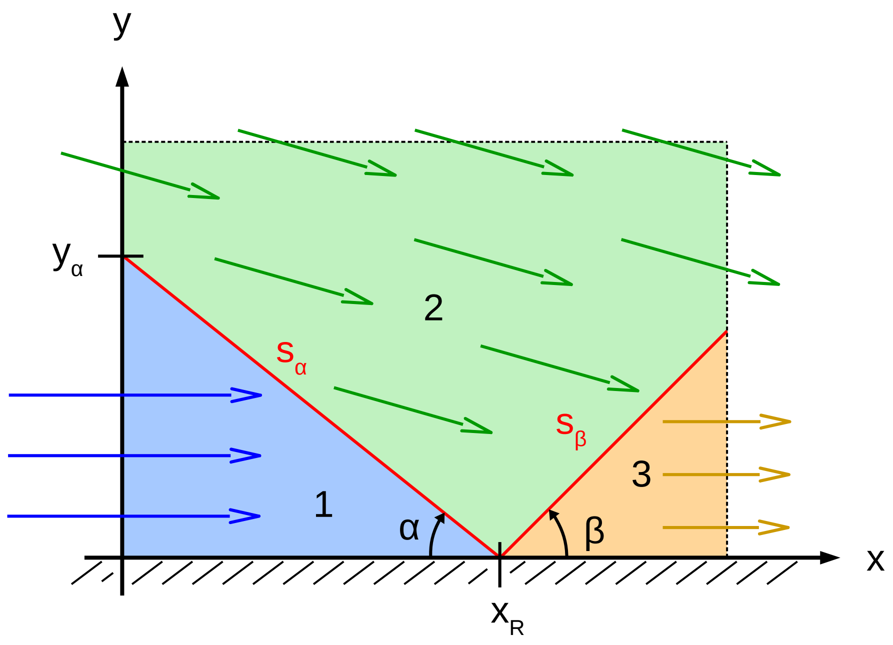


Figure 2: Sketch of the regular shock reflection test case. The colors blue, green and orange denote regions 1, 2 and 3 respectively. The red lines represent the shocks. The colored arrows represent the flow velocity vectors.

$$\mathbf{V}(x,y) = \begin{cases} \mathbf{V}_1 & \text{for} \quad x < x_R \quad \text{and} \quad y < s_\alpha(x) \\ \mathbf{V}_2 & \text{for} \quad x < x_R \quad \text{and} \quad y > s_\alpha(x) \\ \mathbf{V}_2 & \text{for} \quad x > x_R \quad \text{and} \quad y > s_\beta(x) \\ \mathbf{V}_3 & \text{for} \quad x > x_R \quad \text{and} \quad y < s_\beta(x) \end{cases} \tag{29}$$

where subscripts 1-3 denote constant values in regions 1-3 respectively.

Figure 2 also shows that the velocities in regions 1 and 3 are parallell with the wall. This is necessary to respect the principle of impermeability, and implies that

$$v_1 = v_3 = 0 \quad . \tag{30}$$

Furthermore, the magnitudes of the velocities decrease when going from region 1 to region 2, and from region 2 to region 3. I.e. the flow slows down when crossing the shocks. This is because the shocks are irreversibilities that cause the entropy to increase and transform some kinetic energy into internal energy. This means that the density and pressure will increase when the fluid passes through the shocks. Summarized we have:

$$\rho_1 < \rho_2 < \rho_3 \quad , \tag{31}$$

$$u_1 > u_2 > u_3 \quad , \tag{32}$$

$$v_2 < v_1 = v_3 = 0 \quad , \tag{33}$$

$$p_1 < p_2 < p_3 \quad , \tag{34}$$

$$M_1 > M_2 > M_3 \quad . \tag{35}$$

So far, we have described the steady state of this test case. The full test case includes the development of this steady state, from the initial conditions (IC),

$$\mathbf{V}(x, y, t = 0) = \begin{cases} \mathbf{V}_1 & \text{for} \quad y \le y_\alpha \\ \mathbf{V}_2 & \text{for} \quad y > y_\alpha \end{cases} \quad , \tag{36}$$

where the constant values in regions 1 and 2 of the steady-state solution are given in the lower and upper part, respectively. The intersection between the incoming shock s_α and the left boundary will be at $y = y_\alpha$ in the IC and throughout the development of the flow. This is dictated by the Boundary conditions (BC), which will be discussed next.

At the left and upper boundaries, the primitive variables are set using the values in regions 1 and 2:

$$\mathbf{V}(x = 0, y, t) = \begin{cases} \mathbf{V}_1 & \text{for} \quad y \le y_\alpha \\ \mathbf{V}_2 & \text{for} \quad y > y_\alpha \end{cases} \quad , \tag{37}$$

$$\mathbf{V}(x, y = y_{max}, t) = \mathbf{V}_2 \quad . \tag{38}$$

As already mentioned, we prescribe impermeability at the lower boundary:

$$v(x, y = 0, t) = 0 \tag{39}$$

At the right boundary we have outflow and therefore prescribe homogeneous Neumann conditions,

$$\left.\frac{\partial \mathbf{V}}{\partial x}\right|_{x = x_{max}} = 0 \quad . \tag{40}$$

These boundary conditions are fully compatible with both the initial conditions (36) and the steady state (29). The development from the IC towards the steady state will be discussed in more detail, and shown by results, in subsection 6.1.

The analytical solution will now be derived for the steady state of the regular shock reflection. It will be used for verification of the numerical solutions to be introduced later in this report. Additionally, parts of the analytical solution will be used to set the boundary and initial conditions.

Based on the Mach number M_1 of the incoming uniform supersonic flow and the angle α in Figure 2, we will find the region specific flow variables (31) - (35). We will use the oblique shock relations,

$$\frac{\hat{w}_n}{w_n} = \frac{\rho}{\hat{\rho}} = 1 - \frac{2}{\gamma+1}\left(1 - \frac{1}{M^2 sin^2(\theta)}\right) \quad , \tag{41}$$

$$\frac{\hat{p}}{p} = 1 + \frac{2\gamma}{\gamma+1}\left(M^2 sin^2(\theta) - 1\right) \quad , \tag{42}$$

$$\hat{w}_t = w_t = w\,cos(\theta) \quad , \tag{43}$$

$$w_n = w\,sin(\theta) \quad , \tag{44}$$

$$sin^{-1}\left(\frac{1}{M}\right) \leq \theta \leq \frac{\pi}{2} \quad , \tag{45}$$

which were found in chapter 7 of [16], and the additional geometric relations,

$$\hat{w} = \sqrt{\hat{w}_n^2 + \hat{w}_t^2} \quad , \tag{46}$$

$$\hat{\theta} = tan^{-1}\left(\frac{\hat{w}_n}{\hat{w}_t}\right) \quad , \tag{47}$$

$$\varphi = \theta - \hat{\theta} \quad . \tag{48}$$

The variables in equations (41)-(48) are visualized in Figure 3 and defined in Table 1.

The lower limit for θ in (45) ensures that we get $\hat{w}_n \leq w_n$ in (41). This is a physical criterion, based on the Rankine-Hugoniot conditions. The upper limit $\frac{\pi}{2}$ defines a normal shock as limiting case. The inequalities (45) imply that supersonic flow $M > 1$ is required for an oblique shock.

We start in region 1, which is shown in Figure 2. We prescribe ρ_1, u_1, v_1 and the Mach number M_1 as input. The pressure p_1 is found using the Mach number and equations (11), (12) and (13):

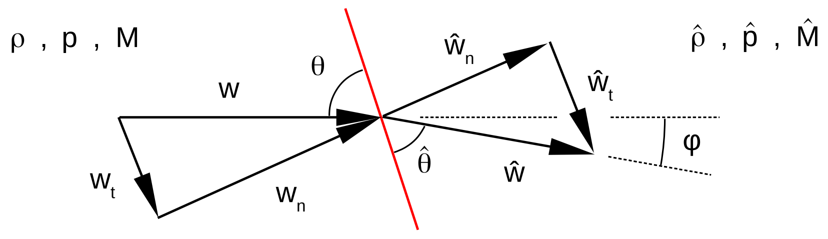


Figure 3: Visual representation of the oblique shock relations (41)-(43), and the relations (44)-(48). The red line represents the oblique shock. The symbols are explained in Table 1.

Table 1: Variable definitions for equations (41)-(48) and Figure 3.

$\mathbf{w}$, $\hat{\mathbf{w}}$	:	Velocity vector $\begin{bmatrix} u \\ v \end{bmatrix}$, upstream and downstream of the shock, respectively.
w , $\hat{w}$	:	Modulus of velocity $\mathbf{w}$ and $\hat{\mathbf{w}}$, upstream and downstream of the shock, respectively.
w_t , $\hat{w}_t$	:	Tangential component of $\mathbf{w}$ and $\hat{\mathbf{w}}$, respectively. Parallell with the shock.
w_n , $\hat{w}_n$	:	Normal component of $\mathbf{w}$ and $\hat{\mathbf{w}}$, respectively. Perpendicular to the shock.
θ , $\hat{\theta}$	:	Smallest angle between the shock and velocity $\mathbf{w}$ and $\hat{\mathbf{w}}$, respectively.
φ	:	Deflection angle. Smallest angle between $\mathbf{w}$ and $\hat{\mathbf{w}}$.
ρ , $\hat{\rho}$	:	Density upstream and downstream of the shock, respectively.
p , $\hat{p}$	:	Pressure upstream and downstream of the shock, respectively.
M , $\hat{M}$	:	Mach number upstream and downstream of the shock, respectively.

$$M_1 = \frac{w_1}{c_1} = \frac{\sqrt{u_1^2 + v_1^2}}{\sqrt{\gamma \frac{p_1}{\rho_1}}} = \frac{\sqrt{u_1^2}}{\sqrt{\gamma \frac{p_1}{\rho_1}}} \quad , \tag{49}$$

where w_1 is the velocity modulus and c_1 is the speed of sound in region 1. By squaring (49) and solving for p_1, we get

$$p_1 = \frac{\rho_1 u_1^2}{\gamma M_1^2} \quad . \tag{50}$$

The angle α, between the incoming shock s_α and the lower boundary wall, is also given as input. Since $v_1 = 0$, we know that $w_1 = u_1$, and that the shock angle is

$$\theta_\alpha = \alpha \quad , \tag{51}$$

as demonstrated in Figure 4. We recall the limits for the shock angle (45), and prescribe α such that

$$sin^{-1}\left(\frac{1}{M_1}\right) \leq \theta_\alpha = \alpha \leq \frac{\pi}{2} \quad . \tag{52}$$

Then, θ_α is used to decompose the velocity vector $\mathbf{w}_1$ into

$$w_{t\alpha} = w_1 cos\left(\theta_\alpha\right) \quad , \tag{53}$$

$$w_{n\alpha} = w_1 sin\left(\theta_\alpha\right) \quad , \tag{54}$$

where $w_{t\alpha}$ is the component of $\mathbf{w}_1$ that is *tangential* to s_α, and $w_{n\alpha}$ is the *normal* component. This decomposition can be seen in the left part of Figure 4.

When the flow passes through the oblique shock s_α, (43) yields that the tangential velocity component is unchanged. I.e.

$$\hat{w}_{t\alpha} = w_{t\alpha} \quad , \tag{55}$$

where $\hat{w}_{t\alpha}$ is the tangential velocity component behind the shock. On the contrary, by applying (41) the normal velocity component will be decreased:

$$\frac{\hat{w}_{n\alpha}}{w_{n_\alpha}} = 1 - \frac{2}{\gamma + 1}\left(1 - \frac{1}{M_1^2 sin^2(\theta_\alpha)}\right) \quad . \tag{56}$$

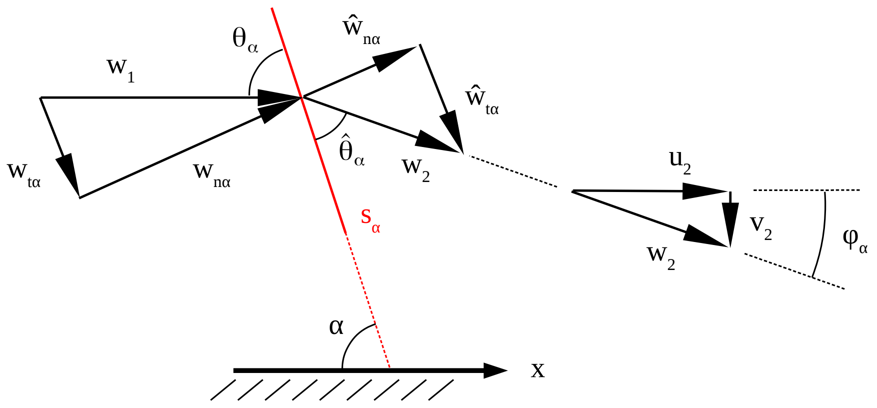


Figure 4: Velocity components and angles that were used to find u_2 and v_2. Subscripts 1 and 2 denote variables that are constant throughout region 1 and 2, respectively. Subscript α denotes variables related to the incoming shock s_α. The hat ^ denotes a variable on the downstream side of the shock.

Solving (56) for $\hat{w}_{n\alpha}$ yields

$$\hat{w}_{n\alpha} = w_{n_\alpha}\left[1 - \frac{2}{\gamma+1}\left(1 - \frac{1}{M_1^2 sin^2(\theta_\alpha)}\right)\right] \quad . \tag{57}$$

Knowing both the components $\hat{w}_{t\alpha}$ and $\hat{w}_{n\alpha}$, we can use (46) to find the modulus w_2 of the velocity in region 2:

$$w_2 = \sqrt{\hat{w}_{t\alpha}^2 + \hat{w}_{n\alpha}^2} \quad . \tag{58}$$

This relation (58) can also be seen in Figure 4. The right part of the figure shows that the Cartesian components u_2 and v_2 can be computed from w_2, and the deflection angle φ_α. Using (48), the deflection angle can be found as

$$\varphi_\alpha = \theta_\alpha - \hat{\theta}_\alpha \quad , \tag{59}$$

where

$$\hat{\theta}_\alpha = tan^{-1}\left(\frac{\hat{w}_{n\alpha}}{\hat{w}_{t\alpha}}\right) \tag{60}$$

is the angle between the shock s_α and the velocity vector $\mathbf{w}_2$, behind the shock. After φ_α is computed using (59), the Cartesian velocity components, u_2 and v_2 in region 2 can be found as

$$u_2 = w_2 cos(\varphi_\alpha) \quad \text{and} \quad v_2 = -w_2 sin(\varphi_\alpha) \quad . \tag{61}$$

Note the minus sign when computing v_2 in (61). The minus is there, because φ_α will always be positive when θ_α is limited as in (52).

Recalling (41) and applying the notation for regions 1 and 2, we find the density ρ_2 in region 2:

$$\begin{aligned} \frac{\rho_1}{\rho_2} &= 1 - \frac{2}{\gamma+1}\left(1 - \frac{1}{M_1^2 sin^2(\theta_\alpha)}\right) \\ \rho_2 &= \frac{\rho_1}{1 - \frac{2}{\gamma+1}\left(1 - \frac{1}{M_1^2 sin^2(\theta_\alpha)}\right)} \end{aligned} \tag{62}$$

where

$$\rho_2 \geq \rho_1 \quad \text{because} \quad \left[\quad \hat{w}_{n\alpha} \leq w_{n\alpha} \quad \text{and} \quad \frac{\hat{w}_{n\alpha}}{w_{n\alpha}} = \frac{\rho_1}{\rho_2} \quad \right] \tag{63}$$

The pressure p_2 in region 2 is found using (42):

$$\frac{p_2}{p_1} = 1 + \frac{2\gamma}{\gamma+1}\left(M_1^2 sin^2(\theta_\alpha) - 1\right) \quad , \tag{64}$$

$$p_2 = p_1\left[1 + \frac{2\gamma}{\gamma+1}\left(M_1^2 sin^2(\theta_\alpha) - 1\right)\right] \quad . \tag{65}$$

Knowing the density ρ_2 and pressure p_2 in region 2, we find the speed of sound c_2 in this region as well, using (11) from section 2:

$$c_2 = \sqrt{\gamma\frac{p_2}{\rho_2}} \quad . \tag{66}$$

The Mach number M_2 in region 2 can then be found as

$$M_2 = \frac{w_2}{c_2} \quad . \tag{67}$$

At this state we can express all the primitive variables and the Mach numbers in regions 1 and 2. We will now use the oblique shock relations again, with the variables from region 2, to obtain

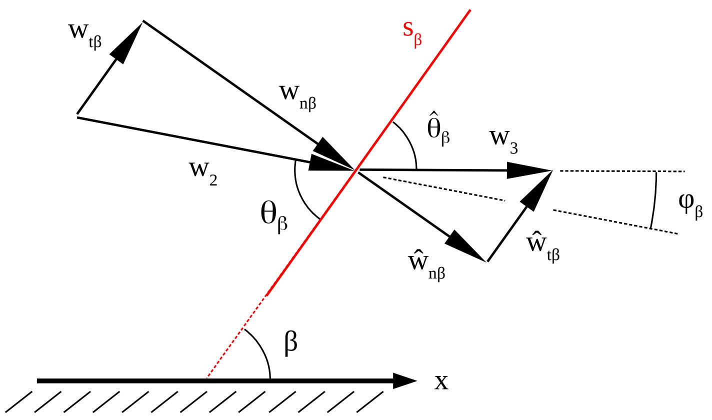


Figure 5: Velocity components and angles that are used to find the deflection angle φ_β. Subscripts 2 and 3 denote variables that are constant throughout regions 2 and 3, respectively. Subscript β denotes variables related to the reflected shock s_β. The hat ^ denotes variables on the downstream side of the shock.

the values for region 3. However, for the reflected shock s_β we do not know the angles β or θ_β, which are shown in Figure 5. Instead, we know that the y-component v_3 of the velocity in region 3 must be

$$v_3 = 0 \tag{68}$$

for the flow to become parallell with the wall after crossing s_β. This means that we cannot derive explicit expressions for the variables in region 3. Instead we will solve this iteratively.

Similar to the incoming shock s_α, we need to ensure $\hat{w}_{n\beta} \leq w_{n\beta}$ at the reflected shock s_β. Therefore, we limit our guesses of θ_β in the iterative solution as in (45):

$$sin^{-1}\left(\frac{1}{M_2}\right) \leq \theta_\beta \leq \frac{\pi}{2} \quad . \tag{69}$$

Figure 5 also shows how the guessed angle θ_β of the reflected shock is used to decompose $\mathbf{w}_2$ into tangential and normal components,

$$w_{t\beta} = w_2 cos(\theta_\beta) \quad , \tag{70}$$

$$w_{n\beta} = w_2 sin(\theta_\beta) \quad . \tag{71}$$

As for the incoming shock, the tangential component remains unchanged through the shock,

$$\hat{w}_{t\beta} = w_{t\beta} \quad . \tag{72}$$

The normal component is decreased, by using relation (41) with the Mach number M_2 from region 2 and the guessed θ_β:

$$\hat{w}_{n\beta} = w_{n_\beta}\left[1 - \frac{2}{\gamma+1}\left(1 - \frac{1}{M_2^2 sin^2(\theta_\beta)}\right)\right] \quad . \tag{73}$$

We do not compute the velocity modulus w_3 before the correct angle θ_β is determined by the iterative method. Instead we find the angle

$$\hat{\theta}_\beta = tan^{-1}\left(\frac{\hat{w}_{n\beta}}{\hat{w}_{t\beta}}\right) \quad , \tag{74}$$

which is also shown in Figure 5. Taking one more look at Figure 5, we see that the angle between the shock and the wall is

$$\beta = \hat{\theta}_\beta \quad . \tag{75}$$

We can also see that the new deflection angle can be found as

$$\varphi_\beta = \theta_\beta - \hat{\theta}_\beta \quad . \tag{76}$$

Again, limiting θ_β in (69) causes the deflection angle φ_β to be positive for all valid input. Knowing this, and comparing φ in Figure 4 and Figure 5, we see that the deflection angles φ_α and φ_β must be equal, for the flow in region 3 to become parallell with the wall. Mathematically:

$$\left[\quad v_1 = 0 \quad \text{and} \quad v_3 = 0 \quad\right] \quad \Rightarrow \quad \varphi_\alpha = \varphi_\beta \quad . \tag{77}$$

Therefore, we keep guessing θ_β and computing φ_β until $\varphi_\alpha = \varphi_\beta$. To guess systematically, the secant method is used. That is, the next value to guess is found by

$$\theta_\beta^m = \theta_\beta^{m-1} - \left(\varphi_\beta^{m-1} - \varphi_\alpha\right)\frac{\theta_\beta^{m-1} - \theta_\beta^{m-2}}{\varphi_\beta^{m-1} - \varphi_\beta^{m-2}} \quad , \tag{78}$$

where m denotes the iteration number. Since the iterative scheme (78) uses the two previous iterations $m-1$ and $m-2$, we need to supply two initial guesses. The initial guesses are set inside the limits given by (69). The secant method (78) is then applied until

$$|\varphi_\alpha - \varphi_\beta| < 10^{-10} \quad . \tag{79}$$

When that happens, θ_β, φ_β, $\hat{w}_{n\beta}$ and $\hat{w}_{t\beta}$ are accepted as valid. Knowing the normal and tangential velocity components behind the shock, the absolute value w_3 of the velocity in region 3 can be found from relation (46),

$$w_3 = \sqrt{\hat{w}_{n\beta}^2 + \hat{w}_{t\beta}^2} \quad , \tag{80}$$

as seen in Figure 5.

Since we postulated $v_3 = 0$, we have flow parallel with the wall, and

$$u_3 = w_3 \quad . \tag{81}$$

The shock angle θ_β, that was found iteratively, is then used to compute the density and pressure in region 3, applying relations (41) and (42), respectively:

$$\rho_3 = \frac{\rho_2}{1 - \frac{2}{\gamma+1}\left(1 - \frac{1}{M_2^2 sin^2(\theta_\beta)}\right)} \quad , \tag{82}$$

$$p_3 = p_2 \left[1 + \frac{2\gamma}{\gamma+1}\left(M_2^2 sin^2(\theta_\beta) - 1\right)\right] \quad . \tag{83}$$

Using ρ_3 and p_3 in equation (11) gives the speed of sound in region 3,

$$c_3 = \sqrt{\gamma \frac{p_3}{\rho_3}} \quad , \tag{84}$$

which in turn gives us the Mach number,

$$M_3 = \frac{w_3}{c_3} \quad . \tag{85}$$

The iterative method (78) used to solve region 3 can converge towards two different solutions. In one of the solutions the reflected shock is a weak shock, which gives supersonic flow $M_3 > 1$ in region 3. In the other solution, the reflected shock is a strong shock, giving a larger angle β and subsonic flow $M_3 < 1$ in region 3. The numerical solutions always include a weak shock and supersonic flow in region 3. To help the secant method find the weak shock solution we choose the initial guesses for θ_β in (78) closer to the lower limit in (69) than the upper limit. Weighted averages of said limits give the initial guesses,

$$\theta_\beta^0 = 0.8\, tan^{-1}\left(\frac{1}{M_2}\right) + 0.2\,\frac{\pi}{2} \quad ,$$

$$\theta_\beta^1 = 0.6\, tan^{-1}\left(\frac{1}{M_2}\right) + 0.4\,\frac{\pi}{2} \quad ,$$

which never produced the strong shock solution.

At this stage, we have obtained the primitive variables and the Mach numbers in all the regions of the flow. The last part of finding the analytical solution is determining the spatial extents of the regions. The regions' borders are set by the domain boundaries,

$$0 \le x \le x_{max} \quad \text{and} \quad 0 \le y \le y_{max} \quad , \tag{86}$$

and the shock characteristics which we will find next. To do this we prescribe y_α, the y-coordinate where s_α intersects the left boundary $x = 0$, as shown in Figure 2. The incoming shock characteristic will be

$$s_\alpha(x) = y_\alpha - tan(\alpha)x \quad . \tag{87}$$

We also see in Figure 2 that the reflection point is found by

$$\begin{aligned} tan(\alpha) &= \frac{y_\alpha}{x_R} \quad , \\ x_R &= \frac{y_\alpha}{tan(\alpha)} \quad . \end{aligned} \tag{88}$$

Furthermore, Figure 2 shows that the reflected shock characteristic will be

$$s_\beta(x) = (x - x_R)tan(\beta) \quad , \tag{89}$$

because

$$s_\beta(x_R) = 0 \quad \text{and} \quad \frac{\partial s_\beta}{\partial x} = tan(\beta) \quad . \tag{90}$$

Then we have all the values that comprise the analytical steady-state solution of the regular oblique shock reflection (29),

Table 2: Numerical values for the analytical solution of the shock reflection test case. The highlighted blue values were prescribed as input.

(a) Global values

x_{max}	1.0
y_{max}	0.6
y_α	0.4
x_R	0.5506
α	$\frac{\pi}{5}$
β	$\frac{\pi}{5.576}$

(b) Region-specific values

	Region 1	Region 2	Region 3
ρ	1.00	1.810	3.009
u	1.00	0.8454	0.6895
v	0.00	-0.2128	0.00
p	0.1143	0.2689	0.5564
M	2.50	1.912	1.397

$$\mathbf{V}(x,y) = \begin{cases} \mathbf{V}_1 & \text{for} \quad x < x_R \quad \text{and} \quad y < s_\alpha(x) \\ \mathbf{V}_2 & \text{for} \quad x < x_R \quad \text{and} \quad y > s_\alpha(x) \\ \mathbf{V}_2 & \text{for} \quad x > x_R \quad \text{and} \quad y > s_\beta(x) \\ \mathbf{V}_3 & \text{for} \quad x > x_R \quad \text{and} \quad y < s_\beta(x) \end{cases} \quad . \tag{91}$$

The numerical values that were prescribed, and the resulting computed variables are listed in Table 2. By setting ρ_1 and u_1 in region 1 to

$$\rho_1 = u_1 = 1 \quad , \tag{92}$$

as in Table 2, we implicitly give the density scale ρ_∞ and the velocity scale u_∞, used in the non-dimensionalization (14). ρ_∞ will then be the dimensional density $[{}^{kg}/{}_{m^3}]$ in region 1, and u_∞ is the magnitude of the dimensional velocity $[{}^{m}/{}_{s}]$ in region 1.

In this report, all the results from the shock reflection test case were obtained using $M_1 = 2.5$, as seen in Table 2. However, M_1 can be varied as a parameter, to give variations of the test case, as long as it is set large enough to give supersonic flow in all regions:

$$M_2 > M_3 > 1 \quad . \tag{93}$$

As mentioned, α is prescribed to satisfy the limits in (52). The chosen value $\alpha = \frac{\pi}{5}$ is much closer to the lower limit than the upper. The reason is that higher values of α caused the regular shock reflection to transition into a Mach reflection in the numerical solutions. Mach reflections are, however, outside the scope of this report.

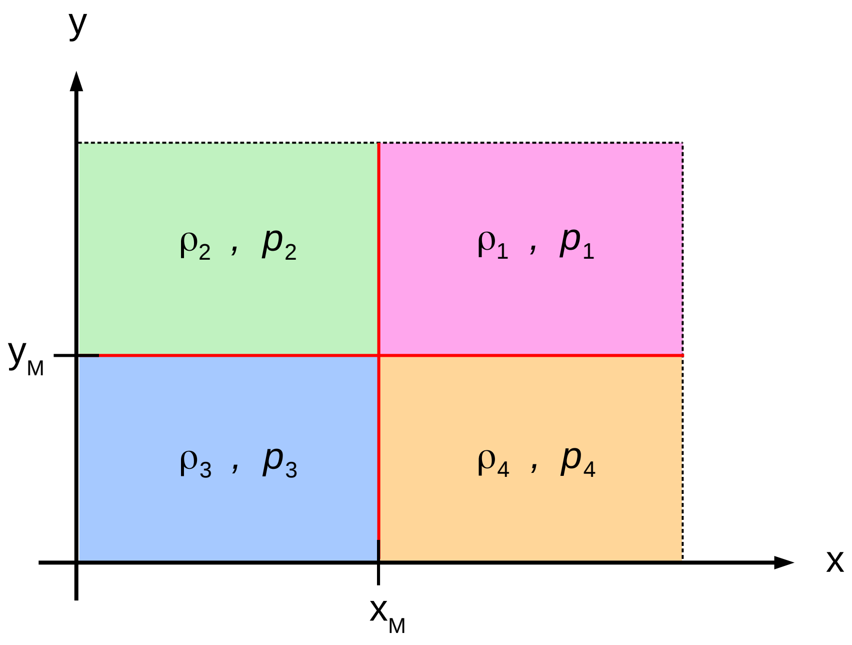


Figure 6: Initial conditions for the 2D Riemann problem. The red lines represent diaphragms.

3.2 2D Riemann Problem

The second test case that was studied is a two-dimensional (2D) Riemann problem. This is a rather theoretical flow, which is difficult to achieve in a physical experiment. It features shocks, contact discontinuities and rarefaction waves of different magnitudes and angles. Thus, it provides insight on the interactions between these different types of waves.

The flow domain is divided into 4 regions, as illustrated in Figure 6. The initial condition is that the density and pressure are constant inside the regions, but differ from one region to another. We envision this being achieved by thin diaphragms separating the regions at x_M and y_M, as in Figure 6. We prescribe the following numerical values for the diaphragm positions and the domain boundaries:

$$x_M = y_M = 0.5 \quad , \tag{94}$$

$$0 \leq x \leq 1 \quad , \tag{95}$$

$$0 \leq y \leq 1 \quad . \tag{96}$$

Thus, the IC is

$$\mathbf{V}(x,y,t=0) = \begin{cases} \mathbf{V}_1 & \text{for} \quad x > x_M \quad \text{and} \quad y > y_M \\ \mathbf{V}_2 & \text{for} \quad x < x_M \quad \text{and} \quad y > y_M \\ \mathbf{V}_3 & \text{for} \quad x < x_M \quad \text{and} \quad y < y_M \\ \mathbf{V}_4 & \text{for} \quad x > x_M \quad \text{and} \quad y < y_M \end{cases} \quad , \tag{97}$$

where subscripts 1-4 denotes the constant values in regions 1-4 respectively. The fluid is initially still in all the regions. I.e.,

$$u(x,y,t=0) = v(x,y,t=0) = 0 \quad . \tag{98}$$

The densities in (97) are prescribed as

$$\rho_1 = 4 \quad , \quad \rho_2 = 3 \quad , \quad \rho_3 = 1 \quad , \quad \rho_4 = 2 \quad , \tag{99}$$

which means that the density scale ρ_∞ in (14) is the dimensional density in region 3, at $t = 0$. Since the initial velocity is zero, the pressures cannot be set using the Mach numbers as we did in (49). Instead we reorder the ideal gas law (6) as

$$\frac{p}{\rho} = RT \quad . \tag{100}$$

If the initial temperature $T(x,y,t=0)$ is constant, then

$$\frac{p_1}{\rho_1} = \frac{p_2}{\rho_2} = \frac{p_3}{\rho_3} = \frac{p_4}{\rho_4} \quad . \tag{101}$$

To ensure that (101) holds, we can prescribe the pressures in (97) as

$$p_1 = 4 \quad , \quad p_2 = 3 \quad , \quad p_3 = 1 \quad , \quad p_4 = 2 \quad , \tag{102}$$

which implies that the reference pressure is the dimensional pressure in region 3, at $t = 0$.

The boundaries in this test case are artificial, not walls. Physically, we picture this as if our frame of reference is close to the diaphragm intersection, and we imagine that the regions and diaphragms extend far beyond this frame. This means that end effects are neglected, and we will only observe effects from the region interfaces. Thus, we specify homogeneous Neumann conditions,

$$\left.\frac{\partial \mathbf{V}}{\partial x}\right|_{x=0} = \left.\frac{\partial \mathbf{V}}{\partial x}\right|_{x=x_{max}} = \left.\frac{\partial \mathbf{V}}{\partial y}\right|_{y=0} = \left.\frac{\partial \mathbf{V}}{\partial y}\right|_{y=y_{max}} = 0 \quad , \tag{103}$$

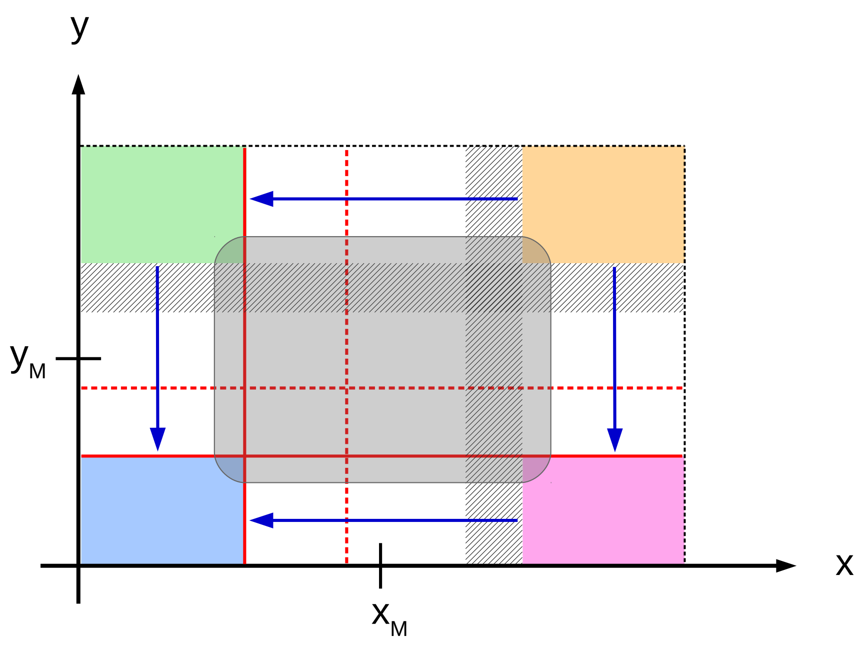


Figure 7: Sketch of the different waves that appear in the 2D Riemann problem. The solid red lines represent shocks, the dashed red lines represent contact discontinuities and the hatched areas represent rarefaction waves. In the colored areas the flow is still equal to the IC. The blue arrows show the direction of the velocity, but *not* the magnitude. The gray rounded rectangle encloses the central area where 2D effects will appear. In the central area, the wavefronts will be nonlinear, even though they are drawn linear in this simplified sketch.

for all the boundaries in this case.

At the time $t = 0$, the diaphragms are removed instantaneously, giving the regions freedom to interact. The gas immediately starts flowing from the higher pressure regions towards the lower pressure regions. Figure 7 shows how shocks and contact discontinuities are propagating into the lower pressure regions, and rarefaction waves are propagating into the higher pressure regions.

Additionally, Figure 7 shows how the initial conditions are retained in the areas where none of the waves have passed. These areas become smaller as the waves propagate. Without viscosity, energy dissipation or obstacles, the waves would proceed forever. However, we will limit the test case to the time intervall before any of the waves reach the artificial bundaries.

Another observation we make from Figure 7 is that there are 4 *one*-dimensional Riemann problems along the boundaries. They are located between the corner regions where the IC is retained. I.e. the same areas as the blue arrows in Figure 7. This means that all the 2D effects will be confined to the central area of the domain, which is represented by the gray rounded rectangle in Figure 7. In the central area, the interactions between the different waves cause the wavefronts to become nonlinear, as the results will show in subsection 6.3.

4 Numerical Approach

In this section the Numerical method will be outlined.

4.1 Basis for the Finite Volume Method

The integral form of the Euler equations (22) can be written as

$$\int_{\Omega} \frac{\partial \mathbf{U}}{\partial t} \mathrm{d}V + \int_{\partial\Omega} \left[\underline{F}(\mathbf{U})\, \mathbf{n} \right] \mathrm{d}A = 0 \quad , \tag{104}$$

where Ω is a control volume with surface $\partial\Omega$. $\underline{F} = [\mathbf{F}(\mathbf{U}), \mathbf{G}(\mathbf{U})]$ is a matrix where the columns are equal to the flux vectors (24) and $\mathbf{n}$ is the normal vector of the surface $\partial\Omega$. We use 2D Cartesian grids where the control volumes Ω correspond to rectangular cells i whose surfaces are divided into faces k, as shown in figures 8 and 9. Figure 8a shows that the normal vectors of the faces point away from the cell in focus. That is,

$$\mathbf{n}_{k,e} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad , \quad \mathbf{n}_{k,w} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \quad , \quad \mathbf{n}_{k,n} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbf{n}_{k,s} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \tag{105}$$

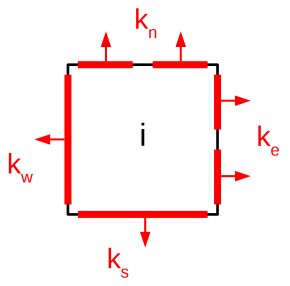
are the normal vectors of faces on the left side, on the right side, below and above the cell i, respectively. The meshes are adaptable by quadrisection with a grading degree $q = 1$, as shown in Figure 1. Thus, each cell will have minimum 4 and maximum 8 enclosing faces, as illustrated in Figure 9. With these mesh specifications in mind, we can write the surface integral in (104) as sums over faces:

$$\int_{\partial\Omega} \left[\underline{F}(\mathbf{U})\, \mathbf{n} \right] \mathrm{d}A = \sum_{k \in k_e} \hat{\mathbf{F}}_k A_k - \sum_{k \in k_w} \hat{\mathbf{F}}_k A_k + \sum_{k \in k_n} \hat{\mathbf{G}}_k A_k - \sum_{k \in k_s} \hat{\mathbf{G}}_k A_k \quad , \tag{106}$$

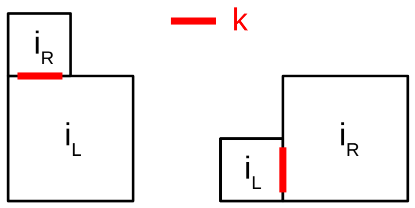
where $\hat{\mathbf{F}}_k(t)$ and $\hat{\mathbf{G}}_k(t)$ are the flux functions $\mathbf{F}(\mathbf{U}(x, y, t))$ and $\mathbf{G}(\mathbf{U}(x, y, t))$ (24), averaged over a face k with area A_k. The subscripts e, w, n, s and k are explained in Figure 8a. We introduce cell averages,

$$\hat{\mathbf{U}}_i(t) = \frac{1}{V_i} \int_{i} \mathbf{U}(x, y, t) \mathrm{d}V \quad , \tag{107}$$

for the conserved variables $\mathbf{U}$ in cell i. Differentiating (107) in time and reordering yields



(a) The cell i is in focus. It has 6 enclosing faces, whose normal vectors point away from the cell i. The subscripts e, w, n and s denote enclosing faces on the right, left, upper and lower side of the cell i.



(b) Two examples where a face k is in focus. Its adjacent cells are denoted by subscripts L and R. For vertical faces the subscripts denote cells on the left and right side of the face, respectively. For horizontal faces the subscripts L and R denote cells below and above the face, respectively.

Figure 8: Notation for cells and faces. Variables in cells are denoted by subscript i, L or R, depending on focus. Values at faces are denoted by subscript k.

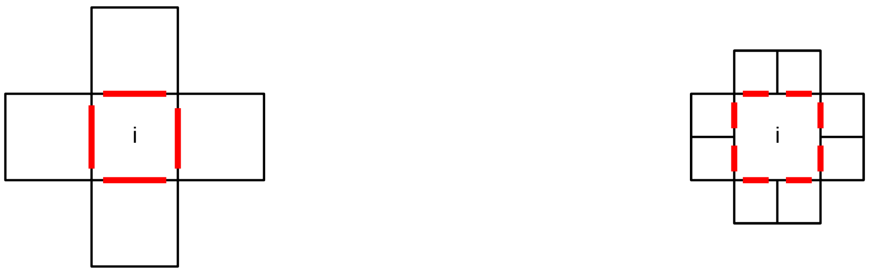
$$\frac{\mathrm{d}\hat{\mathbf{U}}_i(t)}{\mathrm{d}t} V_i = \int_i \frac{\partial \mathbf{U}(x,y,t)}{\partial t} \mathrm{d}V \quad , \tag{108}$$

where $\frac{\partial}{\partial t}$ could enter the integral because the limits of integration are independant of t. The face averaged fluxes $\hat{\mathbf{F}}_k(t)$ and $\hat{\mathbf{G}}_k(t)$, and the cell averaged conserved variables $\hat{\mathbf{U}}_i(t)$ are still exact, but to complete our numerical method we need to approximate them. We let $\mathbf{F}_k(t)$, $\mathbf{G}_k(t)$ and $\mathbf{U}_i(t)$ denote the spatially discretized approximations of the fluxes and conserved variables. Substituting (106) and (108) into (104), using the spatially discreet approximations, yields

$$\frac{\mathrm{d}\mathbf{U}_i}{\mathrm{d}t} V_i = \sum_{k \in k_w} \mathbf{F}_k A_k - \sum_{k \in k_e} \mathbf{F}_k A_k + \sum_{k \in k_s} \mathbf{G}_k A_k - \sum_{k \in k_n} \mathbf{G}_k A_k \quad . \tag{109}$$

(109) is a template for our finite volume method (FVM). To complete it we must choose methods to approximate the fluxes at the faces and to discretize in time.

We set the cell sizes $\Delta x_i = \Delta y_i$, which means that all cells are squares. Physically, this models extrusions, i.e., square prisms where nothing depends on the third spatial dimension. Therefore we refer to V_i and A_k as volumes and areas, respectively, even though they are mathematically areas and lengths in this model.



(a) The cell i has only one enclosing face at each side, giving a total of 4 enclosing faces. This is the lowest possible number.

(b) The cell i has two enclosing faces at each side, giving a total of 8 enclosing faces. This is the highest possible number.

Figure 9: Minimum and maximum number of enclosing faces for a cell i.

4.2 Time Discretization

The approximated fluxes and variables in our FVM template (109) are continuous in time t, but in our numerical method the solutions will only exist at discreet time levels n. We introduce the fully discretized approximations of the fluxes, $\mathbf{F}_k^n$ and $\mathbf{G}_k^n$, and the conserved variables $\mathbf{U}_i^n$, where superscript n denotes the time level.

To approximate the time derivative in (109) we use the first order explicit Euler method,

$$\frac{\mathrm{d}\mathbf{U}_i}{\mathrm{d}t} \approx \frac{\mathbf{U}_i^{n+1} - \mathbf{U}_i^n}{\Delta t^n} \quad . \tag{110}$$

$\Delta t^n = t^{n+1} - t^n$ is the time step size, which is set on every time level n to respect the CFL condition. I.e., we set Δt^n such that the maximum Courant number is

$$C_{max}^n = \frac{\max_i\{|u_i^n| + c_i^n\}\Delta t^n}{\min_i\{\Delta x_i\}} + \frac{\max_i\{|v_i^n| + c_i^n\}\Delta t^n}{\min_i\{\Delta y_i\}} \leq 1 \quad . \tag{111}$$

Introducing this time discretization into the FVM template (109) and reordering, gives the explicit numerical scheme,

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n \frac{\Delta t^n}{V_i} \left(\sum_{k \in k_e} \mathbf{F}_k^n A_k - \sum_{k \in k_w} \mathbf{F}_k^n A_k + \sum_{k \in k_n} \mathbf{G}_k^n A_k - \sum_{k \in k_s} \mathbf{G}_k^n A_k \right) \quad . \tag{112}$$

4.3 Flux Approximation

The fluxes $\mathbf{F}_k^n$ and $\mathbf{G}_k^n$ in (112) are approximated at the vertical and horizontal faces, respectively. For this we use the first order Rusanov method, also known as the local Lax-Friedrichs method.

At face k we have:

$$\begin{aligned}
\mathbf{F}_k^n &= \frac{1}{2}\Big[\Big(\mathbf{F}(\mathbf{U}_R^n) + \mathbf{F}(\mathbf{U}_L^n)\Big) - a_k^n \left(\mathbf{U}_R^n - \mathbf{U}_L^n\right)\Big] \quad , \\
\mathbf{G}_k^n &= \frac{1}{2}\Big[\Big(\mathbf{G}(\mathbf{U}_R^n) + \mathbf{G}(\mathbf{U}_L^n)\Big) - b_k^n \left(\mathbf{U}_R^n - \mathbf{U}_L^n\right)\Big] \quad ,
\end{aligned} \tag{113}$$

where subscripts L, R and k are explained in Figure 8b. The characteristic speeds

$$a_k = \max\{|u_L| + c_L \ , \ |u_R| + c_R\} \quad , \quad b_k = \max\{|v_L| + c_L \ , \ |v_R| + c_R\} \quad , \tag{114}$$

are determined by the spectral radii of the Jacobian matrices $\frac{\partial \mathbf{F}(\mathbf{U})}{\partial \mathbf{U}}$ and $\frac{\partial \mathbf{G}(\mathbf{U})}{\partial \mathbf{U}}$, respectively, in the cells i_L and i_R. $c_i = \sqrt{\gamma p_i/\rho_i}$ is the speed of sound in cell i.

4.4 Numerical Treatment of Initial and Boundary Conditions

4.4.1 Shock Reflection

The left (37) and upper (38) boundary conditions for the shock reflection test case, cf. Figure 2, are passed directly to the analytical flux functions (24). I.e., for a face k in the left boundary we have

$$\mathbf{F}_k^n = \begin{cases} \mathbf{F}(\mathbf{U}(\mathbf{V}_1)) & \text{if} \quad y_k < y_\alpha \\ \mathbf{F}(\mathbf{U}(\mathbf{V}_2)) & \text{if} \quad y_k > y_\alpha \end{cases} \quad , \tag{115}$$

where y_k is the y-coordinate of any point on face k. This means that we always ensure that y_α is at a cell interface, i.e., all cells and vertical faces are either entirely above or below y_α. At a face k in the upper boundary, all the fluxes are computed using variables from region 2:

$$\mathbf{G}_k^n = \mathbf{G}(\mathbf{U}(\mathbf{V}_2)) \quad . \tag{116}$$

The homogeneous Neumann conditions (40) at the right boundary are treated by extrapolating the variables from the left adjacent cell. Thus, for a face k at the right boundary, we have

$$\mathbf{F}_k^n = \mathbf{F}(\mathbf{U}_L) \quad , \tag{117}$$

by the notation in Figure 8b. The impermeability (39) at the lower boundary is treated using the concept of ghost cells, but without adding actual ghost cells in the program. For each face

k in the lower boundary, we imagine that below the face there is a cell, i_L by the notation in Figure 8b. In this cell the primitive variables $\mathbf{V}_L$ are symmetrical with $\mathbf{V}_R$ in the cell above face k. I.e.,

$$\rho_L = \rho_R \quad , \quad u_L = u_R \quad , \quad v_L = -v_R \quad , \quad p_L = p_R \quad , \tag{118}$$

where we notice the minus sign for v. Evaluating the Rusanov flux approximation $\mathbf{G}_k^n$ (113) for our lower boundary face k using $\mathbf{U}_R$ and $\mathbf{U}(\mathbf{V}_L)$ gives

$$\mathbf{G}_k^n = \begin{bmatrix} 0 \\ 0 \\ G_{\rho v,k}^n \\ 0 \end{bmatrix} \quad . \tag{119}$$

$G_{\rho v,k}^n$ is the Rusanov approximation of the vertical momentum density flux,

$$G_{\rho v}(\mathbf{U}) = \rho v^2 + p \quad , \tag{120}$$

which is the third element of the flux vector $\mathbf{G}(\mathbf{U})$ (24). The approximation (113) gives that

$$G_{\rho v,k}^n = \frac{1}{2}\Big[\Big(G_{\rho v}(\mathbf{U}_R^n) + G_{\rho v}(\mathbf{U}_L^n)\Big) - b_k^n \Big([\rho v]_R^n - [\rho v]_L^n\Big)\Big] \quad . \tag{121}$$

We recall the symmetry (118) of the variables, such that

$$\begin{aligned} G_{\rho v,k}^n &= \frac{1}{2}\Big[\Big(\rho_R^n [v_R^n]^2 + p_R^n + \rho_R^n[-v_R^n]^2 + p_R^n\Big) - b_k^n \Big(\rho_R^n v_R^n - \rho_R^n[-v_R^n]\Big)\Big] \\ &= \frac{1}{2}\Big[2\left(\rho_R^n [v_R^n]^2 + p_R^n\right) - \left(|v_R^n| + c_R^n\right)\left(2\rho_R^n v_R^n\right)\Big] \\ &= \rho_R^n [v_R^n]^2 + p_R^n - \rho_R^n v_R^n |v_R^n| - \rho_R^n v_R^n c_R^n \\ &= \rho_R^n v_R^n \left(v_R^n - |v_R^n| - c_R^n\right) + p_R^n \quad , \end{aligned} \tag{122}$$

where (114) was used to express b_k^n. The fluxes $\mathbf{G}_k^n$ at our lower boundary face k are set as (119), where $G_{\rho v,k}^n$ (122) only depends on the variables $\mathbf{U}_R$ in the cell above.

The initial conditions (36) for the shock reflection case are treated by setting the variables,

$$\mathbf{V}_i^0 = \begin{cases} \mathbf{V}_1 & \text{if} \quad y_i < y_\alpha \\ \mathbf{V}_2 & \text{if} \quad y_i > y_\alpha \end{cases} , \tag{123}$$

explicitly in each cell. Again, we note that the meshes are chosen such that all cells are either entirely above y_α or entirely below it.

4.4.2 2D Riemann Problem

The 2D Riemann problem has homogeneous Neumann conditions (103) at all its boundaries. We treat them the same way as the right boundary in the shock reflection case (117), by extrapolating the variables from the only adjacent cell. Thus, the fluxes at the boundary faces k are

$$\mathbf{F}_k^n = \begin{cases} \mathbf{F}(\mathbf{U}_R^n) & \text{if} \quad x_k = 0 \\ \mathbf{F}(\mathbf{U}_L^n) & \text{if} \quad x_k = x_{max} \end{cases} , \tag{124}$$

$$\mathbf{G}_k^n = \begin{cases} \mathbf{G}(\mathbf{U}_R^n) & \text{if} \quad y_k = 0 \\ \mathbf{G}(\mathbf{U}_L^n) & \text{if} \quad y_k = y_{max} \end{cases} , \tag{125}$$

for vertical and horizontal faces, respectively. x_k and y_k are coordinates of any point at face k.

The initial conditions (97) are passed to the cells i such that

$$\mathbf{V}_i^n = \begin{cases} \mathbf{V}_1 & \text{if} \quad x_i > x_M \quad \text{and} \quad y_i > y_M \\ \mathbf{V}_2 & \text{if} \quad x_i < x_M \quad \text{and} \quad y_i > y_M \\ \mathbf{V}_3 & \text{if} \quad x_i < x_M \quad \text{and} \quad y_i < y_M \\ \mathbf{V}_4 & \text{if} \quad x_i > x_M \quad \text{and} \quad y_i < y_M \end{cases} , \tag{126}$$

where x_i and y_i are the coordinates of any point inside cell i. This means that we have taken the same precautions for x_M and y_M as we did for y_α. I.e., the vertical line $x = x_M$ and the horizontal line $y = y_M$ do not bisect any cell i.

4.5 Stopping Criterion for the Steady-State Problem

For the test case with the regular oblique shock reflection, a convergence criterion is needed. When the solution converges towards its steady state, the time derivatives $\frac{\partial[]}{\partial t}$, in the governing equations (1)-(4), will converge towards zero. We assumed that it suffices to demand the rate of density change $\frac{\partial \rho}{\partial t} \to 0$ for the stopping criterion. With the explicit Euler method (110), we have the approximation

$$\left(\frac{\partial \rho}{\partial t}\right)_i^n \approx \frac{\rho_i^{n+1} - \rho_i^n}{\Delta t^n} = \frac{\Delta \rho_i^n}{\Delta t^n} \quad , \tag{127}$$

for $\frac{\partial \rho}{\partial t}$ in cell i. Steady state is assumed at time level $n+1$ if

$$\left\| \frac{\Delta \rho^n}{\Delta t^n} \right\|_1 \leq 10^{-4} \left\| \frac{\Delta \rho^0}{\Delta t^0} \right\|_1 \quad , \tag{128}$$

where

$$\left\| \frac{\Delta \rho^n}{\Delta t^n} \right\|_1 = \sum_i \left[V_i \left| \frac{\Delta \rho_i^n}{\Delta t^n} \right| \right] \tag{129}$$

is the 1-norm of the rate of density change. This corresponds to the sum of the moduli of the rate of mass change in all cells i, from time level n to time level $n+1$. It indicates if the total mass in the domain is changing, and if mass is redistributed within the domain. Since the time step size Δt^n is equal for all cells we extract it from the 1-norm.

$$\left\| \frac{\Delta \rho^n}{\Delta t^n} \right\|_1 = \frac{\|\Delta \rho^n\|_1}{\Delta t^n} \tag{130}$$

will be referred to as the mass redistribution rate and serves as convergence indicator. It is required to be lower equal to its initial value by the factor 10^{-4}, before we assume steady state.

4.6 Numerical Error

The error considered in this report is the volume averaged density error,

$$\varepsilon_\rho = \frac{1}{V_D} \|\rho^{n_{max}} - \hat{\rho}_{exact}\|_1 \quad , \tag{131}$$

where $V_D = \sum\limits_i V_i$ is the domain volume, $\rho^{n_{max}}$ is the final density solution and

$$\hat{\rho}_{exact,i} = \frac{1}{V_i} \int\limits_i \rho_{exact}(x, y)\, \mathrm{d}x\, \mathrm{d}y \tag{132}$$

is the exact average density in cell i. The analytical solution $\rho_{exact}(x, y)$ is given by equation (91) and Table 2 in subsection 3.1. Comparing with the exact cell averages (132) instead of evaluating $\rho_{exact}(x, y)$ in the cell centers, gives a more justified comparison, because the numerical solution approximates cell averages.

The errors ε_u, ε_v, ε_p and ε_M of the velocity components, pressure and Mach number were computed in the same manner as ε_ρ.

4.7 Two Finite Volume Solvers

The numerical method outlined in this section was implemented in two different solvers. The first one is a standard finite volume solver, which uses structured, equidistant grids that cannot be adapted. However, the grid spacing can be varied from one simulation to another. The data structure of this structured solver is based on dynamically allocated arrays. The numerical scheme is implemented as it is formulated in (112). I.e., looping over cells and applying the flux balance in one cell at a time.

The second solver uses unstructured adaptive meshes, as explained in the beginning of subsection 4.1. This AMR solver is outlined in detail in section 5. The numerical scheme (112) is implemented as a loop over faces, where the flux transfer is applied through one face at a time. E.g., for interior vertical faces,

$$\begin{aligned} \mathbf{U}_R^{n+1} &:= \mathbf{U}_R^{n+1} + \mathbf{F}_k^n \quad , \\ \mathbf{U}_L^{n+1} &:= \mathbf{U}_L^{n+1} - \mathbf{F}_k^n \quad , \end{aligned} \tag{133}$$

by the notation in Figure 8b. The $:=$ operator denotes assignment to a computer variable. The fluxes $\mathbf{F}_k^n$ are computed using (113), in the same loop.

The reason for using two different solvers was to provide a justified comparison, that can show the potential overhead from the AMR data structure. An alternative approach would be to use the AMR solver, without adapting, as a basis for comparison.

Table 3: Data member variables of the cell class. The variables are explained in the text.

Double-precision floating points:	
	ρ_i^n , ρu_i^n , ρv_i^n , ρE_i^n
	ρ_i^{n-1} , ρu_i^{n-1} , ρv_i^{n-1} , ρE_i^{n-1}
	u_i^n, v_i^n , p_i^n , c_i^n
	$F_{\rho u}(\mathbf{U}_i^n)$, $F_{\rho v}(\mathbf{U}_i^n)$, $F_{\rho E}(\mathbf{U}_i^n)$, $G_{\rho v}(\mathbf{U}_i^n)$, $G_{\rho E}(\mathbf{U}_i^n)$
	V_i
Integer:	
	L_i
Boolean:	
	r_i , m_i , $\tilde{m}_i$
Iterators to faces:	
	k_{w1} , k_{w2} , k_{n1} , k_{n2} , k_{e1} , k_{e2} , k_{s1} , k_{s2}

5 AMR Approach

As mentioned, an adaptation strategy should specify when, where and how to adapt [7]. This section will answer those three questions, one by one. The adaptation trigger decides *when* to adapt. The flagging and grading loops detect *where* to adapt. In sections 5.5 and 5.6 we explain *how* cells are refined and merged. Before we examine these procedures separately, the data structure of the AMR-solver is outlined.

5.1 Data Structure

The data structure in the AMR-solver is object oriented [18]. The cells and faces in the mesh are objects, i.e. instances of the classes *Cell* and *Face*, respectively. Here, class refers to a user-defined data type that can contain multiple data member variables of different types.

The data members of the cell class are listed in Table 3, where the members are sorted after data type [19]. The table shows that the cells contain all the flow variables as data members. That is, the primitive variables (25), the conserved variables (23) and the speed of sound (11). The conserved variables are stored for the current time level n, and the previous time level $n-1$. The first reason for that is to compute the 1-norm of density change (130), used in the stopping criterion (128) and the adaptation trigger (subsection 5.2). The second reason is that the numerical fluxes are computed and transferred in loops over faces. All of cell i's enclosing faces will use $\mathbf{U}_i^{n-1}$ to compute their fluxes and overwrite $\mathbf{U}_i^n$, cf. equation (133).

$F_{\rho u}$, $F_{\rho v}$, $F_{\rho E}$, $G_{\rho v}$ and $G_{\rho E}$ are fluxes evaluated using the exact flux functions (24) with the flow variables $\mathbf{U}_i^n$ in the cell. These values are stored in the cells to avoid computing them redundantly for the flux approximations (113). Since the density fluxes are already available as momentum densities ρu_i^n and ρv_i^n, they are not stored again. Since $F_{\rho v} = G_{\rho u}$, these fluxes share one single data member. Additionally, the cells store their volumes V_i.

The remaining data members in the cells are not used in the numerical method, but are necessary for the mesh adaptation, as we will see in the following subsections. The refinement level L_i was introduced in subsection 1.1. It relates to the cell size and is stored as an integer. There are also 3 boolean (also known as logical, or binary) data members. These are the refine flag r_i, the merge flag m_i, and the *prevent* merge flag $\tilde{m}_i$. These flags are explained with the flagging procedure in subsection 5.3.

Before listing the last data members, we will introduce three new concepts: *pointers*, *containers* and *iterators* [18]. A pointer is a variable that contains the memory address (aka location) [19] of another variable. It is not a copy of the other variable, but gives access to it. A container, in this report, refers to a data structure that contains multiple objects of the same type. A common example is an array, which will soon be further discussed. An iterator is an object used to navigate, and potentially manipulate, a container. We could write a whole other report about pointers and iterators, but in this data structure they are mainly used in the following way: Pointers and iterators are used to access objects in a container, without copying the objects. Iterators are additionally used to insert and delete objects in containers.

The last data members of the cell class are iterators to the faces that enclose the cell. As we saw in Figure 9 in subsection 4.1, a cell can be enclosed by 4 to 8 faces. Each cell stores 8 iterators to faces, 2 for each side: k_{w1} , k_{w2} , k_{n1} , k_{n2} , k_{e1} , k_{e2} , k_{s1} and k_{s2}. Most cells will have only *one* adjacent face on most of its sides. In these cases both the iterators point to the same face on that side. E.g., for cell i in Figure 8a we have $k_{w1} = k_{w2}$ and $k_{s1} = k_{s2}$, while k_{n1} , k_{n2} , k_{e1} and k_{e2} are unique. This means that some unnecessary memory space is used. We tolerate this, since saving memory capacity is not the priority for this solver.

The face class is much simpler, as it contains only 3 data members: the area A_k of the face, and two pointers to cells i_L and i_R. We keep the notation in Figure 8b, so for vertical faces, i_L and i_R are the left and right adjacent cells. Horizontal faces point to the cells below and above themselves. All interior faces have exactly two adjacent cells, but all boundary faces have exactly one. This is solved by letting one of the pointers in the boundary faces point to the address zero, which is called a null-pointer. E.g., a face at the left boundary will have a null-pointer as its i_L. The faces do not store information about whether they are interior, boundary, vertical or horizontal. We will show how this is solved later.

Knowing all the members of both cells and faces, we can see how the connectivity of the mesh is specified [3]. Connectivity, here refers to how the cells and faces are related to one another.

In this solver, all cells can point to their enclosing faces, and all faces can point to their adjacent cell(s). The numerical method outlined in section 4 only requires *one* of these ways. I.e. we can loop over faces and point to cells, as the present AMR solver does, or we could loop over cells and point to faces. In my previous work [1], I developed a solver where the mesh had no cell-to-face connectivity. This made the mesh adaptation algorithms quite complex and difficult to maintain. In the new solver, developed for this report, cell-to-face connectivity was included. This was to improve code maintainability, and allow less complex algorithms.

The trade-off regarding a complex class versus complex algorithms also applies to spatial coordinates. We see that there are no x- or y-values in either of the classes, meaning that neither cells or faces know their position in the spatial domain. Positions, e.g. cell centers, are not needed for the numerical method or for the mesh adaptation, but will require extra work when adapting, to keep the positions correct. The positions are needed after the simulation, for plotting and comparing with the analytical solution to find the error. For these purposes, the cell positions can be found recursively, using the cells' refinement levels and their ordering in the container. The ordering scheme is discussed with the refining loop in subsection 5.5. Summarized, omitting x and y data members gave a simpler cell class, simpler adaptation procedures, and more complex post-processing.

The cell- and face objects are stored in a doubly linked list [14, 18]. A linked list is a container data structure, where the actual data is stored in nodes. The nodes can be located anywhere in the program's heap memory. This means that they are not necessarily close to each other, or even in order, in memory. Instead, the ordering of the nodes is given by pointers. In a *doubly* linked list, each node contains a pointer to the next node *and* a pointer to the previous node. Special treatment is given at the first and last nodes, depending on implementation. In the scope where the list is declared, we have a pointer to the first node, and a pointer to the last node. Thus, we can traverse (aka iterate) the linked list by following the pointer to the first node, and then follow the pointers forward until we find the last node. In a doubly linked list we can also start at the last node, and traverse it backwards, following the pointers to the previous nodes until we find the first one. In general, we cannot start at any of the nodes between the first and the last. I.e., we do not have random access to the data in a linked list.

An alternative container data structure that was considered is dynamic arrays [18]. In an array, objects are located contiguously in memory. Therefore, the next objects can always be found by offsetting the memory address by the object size. This is faster than reading a pointer. Furthermore, it is probable that the next object is already in cache [19], whose access time is much lower than the main memory. Another advantage of arrays is that we save the memory space that the node pointers in the linked lists require. However, several implementations of dynamic arrays allocate much more memory than they use, for extra capacity, to avoid reallocation on every insertion. The biggest disadvantage of using arrays in the AMR-solver is that adding and removing elements in the middle of the array is very costly. As an example, if object number 100 000 is removed from an array of 200 000 objects, the last 100 000 objects must be left-shifted to fill the gap from the removed object. On the contrary, adding and removing nodes in a linked list is very simple: One object is created or deleted, and two pointers are changed. The arguments for choosing linked lists instead of arrays were:

1. The ability to insert and delete with constant complexity.
2. Pointers to objects in a linked list are still valid after adding/removing, whilst pointers to objects after the added/removed object in an array are invalidated.
3. The cost of loading a pointer to find the next element was assumed to be negligible compared to the other work done on each loop.
4. When looping over faces, the cells are not accessed in the same order as they are stored in their container. This was assumed to limit the gain from spatial locality.
5. Random access is not necessary. All operations on cells and faces are done while traversing.

All the cells are stored in one single linked list. On the other hand, the faces are divided into multiple lists: interior vertical, interior horizontal, and one for each boundary. Thus, the faces do not need to know whether they are vertical or horizontal, or wether they are interior or at a boundary. More importantly, this means that we do not need to check any of that while looping. E.g. when looping over interior vertical faces, we already know that i_L and i_R exist, and that we shall transfer the flux $\mathbf{F}_k^n$, not $\mathbf{G}_k^n$, from i_L to i_R.

5.2 Adaptation Trigger

The adaptation trigger decides *when* to adapt. When the trigger activates, the flagging, grading, refining and merging procedures are run, in that order. The trigger is based on accumulating the mass redistribution rate $\|\Delta\rho^n\|_1 / \Delta t^n$ (130), as defined in subsection 4.5.

The first mesh adaptations are triggered at time level $n_0 = 0$, i.e., based on the initial condition (IC). The number of initial adaptations is equal to the upper limit L_{max} for the refinement level. Thus, if there are large gradients or discontinuities in the IC, the nearby cells will be refined to the highest level before the first time step starts. After an adaptation, say at $n = n_a$, the subsequent values of $\|\Delta\rho^n\|_1 / \Delta t^n$ at every time level n are accumulated until

$$\sum_{n=n_a}^{n_{a+1}-1} \frac{\|\Delta\rho^n\|_1}{\Delta t^n} \geq 5 \, \frac{\|\Delta\rho^0\|_1}{\Delta t^0} \tag{134}$$

Then, mesh adaptation is triggered at time level $n = n_{a+1}$. I.e., we accumulate the mass redistribution since the last adaptation, until the accumulated change is so big that we need to adapt the mesh. This is an idea that I have not yet found in the literature. However, I wanted to test it and it has worked very well in all simulations of both test cases this far. When a solution starts to converge, the number of time levels between mesh adaptations becomes higher and higher, as the results will later show.

The factor 5 above was found by trial and error and may be case specific. However, it proved to be a good value both of the test cases in this project. With values higher than 5, I observed that the wavefronts could propagate out of a highly refined area before the mesh was adapted. With values around 1-2, the mesh is adapted more frequently than necessary, which increases the CPU time.

The implementation of the adaptation trigger is simple. After the first time step is computed, we store the value $5\,\frac{\|\Delta\rho^0\|_1}{\Delta t^0}$. We also declare a variable for the sum $\sum$ in (134) and initialize it to zero. On every time level n, after the fluxes are transferred we compute $\|\Delta\rho^n\|_1 / \Delta t^n$ and add it to the sum. Then, if (134) is true, we set the sum-variable to zero and run the flagging, grading, refining and merging loops. If (134) is false we do nothing.

5.3 Flagging

The flagging loop is the first step of each mesh adaptation. It decides where the mesh will be refined, and where it can be coarsened. To decide this, a criterion based on absolute differences in the flow variables ρ, u and v is used. These 3 variables will be referred to as the indicators.

The flagging loop is a loop over all the interior faces. For each face k we check the absolute difference in all the indicators, between the 2 adjacent cells i_R and i_L.

$$\Delta\rho_k = |\rho_R^n - \rho_L^n| \quad , \quad \Delta u_k = |u_R^n - u_L^n| \quad , \quad \Delta v_k = |v_R^n - v_L^n| \quad , \tag{135}$$

where subscripts k, R and L follow the notation in Figure 8b. By looping over interior faces we check all combinations of neighboring cell pairs exactly once. The differences (135) are compared to the refine thresholds

$$\delta_{r,\rho} = \eta_r\,\Delta\rho_{max} \quad , \quad \delta_{r,u} = \eta_r\,\Delta u_{max} \quad , \quad \delta_{r,v} = \eta_r\,\Delta v_{max} \quad , \tag{136}$$

and the merge thresholds

$$\delta_{m,\rho} = \eta_m\,\Delta\rho_{max} \quad , \quad \delta_{m,u} = \eta_m\,\Delta u_{max} \quad , \quad \delta_{m,v} = \eta_m\,\Delta v_{max} \quad , \tag{137}$$

where

$$\begin{aligned}
\Delta\rho_{max} &= \max_{i,0\le l\le n}\{\rho_i^l\} - \min_{i,0\le l\le n}\{\rho_i^l\} \quad , \\
\Delta u_{max} &= \max_{i,0\le l\le n}\{u_i^l\} - \min_{i,0\le l\le n}\{u_i^l\} \quad , \\
\Delta v_{max} &= \max_{i,0\le l\le n}\{v_i^l\} - \min_{i,0\le l\le n}\{v_i^l\} \quad ,
\end{aligned} \tag{138}$$

are the largest spreads for all the indicators, evaluated over all cells i and all time levels l until n. $\max_{i,0\le l\le n}\{\rho_i^l\}$ is the largest density that has existed in any cell i at any time level l until time level n. Conversely, $\min_{i,0\le l\le n}\{\rho_i^l\}$ is the lowest cell density until time level n. The same applies for the other indicators. This means that the spreads (138) can never decrease, and that they are likely to increase. This is a precaution we take to avoid the thresholds to converge towards zero if any of the indicators converge towards a constant distribution. E.g., if u_i^n becomes almost the same value in all cells, we do not want $\delta_{r,u}$ to go towards zero. Even though the spreads (138) cannot decrease, they can start at zero. If the initial condition for velocity component u_i^0 is the same in all cells (cf. the 2D Riemann problem (97)), then we have $\Delta u_{max} = 0$ initially. To avoid refine thresholds $\delta_r = 0$ (136), which would cause all cells to be refined, we set a lower limit for the thresholds, $\delta_r \ge 10^{-5}$. This is a rudimentary solution, but it works for the test cases in this report.

η_r is a is a multiplier that affects all the refine thresholds (136). It sets the thresholds δ_r as fractions of the largest spreads (138). We will refer to η_r as the refine tolerance. In a similar way, η_m sets the *merge* thresholds δ_m in (137), also as fractions of the largest spreads (138). η_m will be referred to as the merge tolerance. Both these tolerances are given as parameters for the AMR-solver, to adjust the strictness of the refinement criterion.

The comparisons between the absolute differences (135) and the thresholds, (136) and (137), are used to assign the refine flag r, the merge flag m and the prevent merge flag $\tilde{m}$. Since they are boolean values they can be either true or false, which we also denote by 1 and 0, respectively. The refine flag r_L in the left/lower adjacent cell of face k is set by the following scheme:

$$\begin{aligned}
& r_L := \\
& \quad (\Delta\rho_k > \delta_{r,\rho} \;\big|\; \Delta u_k > \delta_{r,u} \;\big|\; \Delta v_k > \delta_{r,v}) \\
& \quad \& \;\; L_L < L_{max} \;\big|\; r_L
\end{aligned} \tag{139}$$

where $:=$ denotes assignment of a computer variable. & and $\big|$ are the binary AND and OR operators, respectively, from boolean algebra. $A \;\&\; B = 1$ if and only if *both* the operands $A = B = 1$, otherwise it is false. $A \;\big|\; B = 1$ if and only if *any* of the operands are true. The inequality signs $<$ and $>$, in this subsection, are conditional operators that return true if the condition is true, and false otherwise. The conditional operators have higher precedence than &, which has higher precedence than $\big|$. I.e.,

$$A \;\big|\; B \;\&\; a < b = A \;\big|\; (B \;\&\; [a < b]) \quad , \tag{140}$$

where A and B are booleans and a and b are real numbers.

With these definitions, we can see that $(\Delta\rho_k > \delta_{r,\rho} \;\big|\; \Delta u_k > \delta_{r,u} \;\big|\; \Delta v_k > \delta_{r,v})$ in (139) will be true if *any* of the indicators have a difference higher than its respective threshold. This reflects that we want to refine the mesh in areas where any of the indicators have high gradients. The results in subsection 6.1 will show the effects of this. The next condition is $L_L < L_{max}$, where L_L is the refinement level of the left/lower cell. This ensures that cells at refinement level $L = L_{max}$ cannot be refined further. In the end of (139), the previous state of r_L is ORed in. This implies that the refine flag can only be given, not removed, in this loop. As we saw in figures 8 and 9, all cells have multiple enclosing faces. If the refine flag was set independently of its previous state, then the enclosing face that comes latest in the loop would decide if the cell were refined or not. With the final OR, we ensure that a cell is refined if *any* of its enclosing faces flag it. For this to work, all the cells must have $r = 0$ when the flagging loop starts.

The absolute differences (135) are then compared with the merge thresholds (137), to set the merge flags m_L and m_R in both the adjacent cells of face k. In the left/lower cell,

$$\begin{aligned} m_L :=& \\ & \Delta\rho_k < \delta_{m,\rho} \;\;\&\;\; \Delta u_k < \delta_{m,u} \;\;\&\;\; \Delta v_k < \delta_{m,v} \\ & \&\;\; L_L > 0 \;\;\&\;\; !\tilde{m}_L \end{aligned} \tag{141}$$

where the exclamation mark ! is the NOT operator (aka invert), which is true if and only if the operand is false. The first 3 conditions in (141) ensure that *all* the indicators must have small differences to allow merge flagging. The next condition $L_L > 0$ ensures that cells cannot be merged past level zero. In the end of (141) we see that m_L cannot be set true if the prevent merge flag $\tilde{m}_L = 1$. The purpose of $\tilde{m}$ is to make sure that a cell can only be merge flagged if all its enclosing faces find small differences. Therefore, $\tilde{m}_L$ is assigned as

$$\tilde{m}_L := \;\; !m_L \;\big|\; \tilde{m}_L \tag{142}$$

on each loop. That is, $\tilde{m}_L$ is assigned true if $m_L = 0$, and cannot be changed from $1 \to 0$ throughout the flagging loop. I.e. for this technique to work, all the cells must have $\tilde{m} = 0$ when the loop starts. We could avoid using $\tilde{m}$ altogether. If all the cells had $m = 1$ before the flagging loop, we could allow m to change from $1 \to 0$, but not $0 \to 1$. This would make sure that merge flags are only given if all the enclosing faces condone it. However, in the current implementation, the latter approach would require an extra list traversal, which is too costly.

The flags r_R, m_R and $\tilde{m}_R$ in the right/above cell are set by the same schemes as r_L (139), m_L (141) and $\tilde{m}_L$ (142), but changing the subscripts $L \to R$.

At the end of the flagging loop we know the following conditions to be true:

- Cells are flagged for refinement if *any* of the indicators have large absolute differences over *any* of the enclosing faces, *and* the cell is not at the maximum level L_{max}.
- Cells are flagged for merging if *all* of the indicators have small absolute differences over *all* of the enclosing faces, *and* the cell is not at the minimum level 0.
- No cell has both flags $r = 1$ and $m = 1$. They have either one of them, or none of them.

5.4 Grading

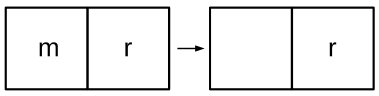
As discussed in subsection 1.1, we will use mesh grading to prevent adjacent cells to be more than 1 refinement level apart. Thus, we require

$$\Delta x_L \in \left\{ \frac{1}{2}\Delta x_R \ , \ \Delta x_R \ , \ 2\,\Delta x_R \right\} \quad , \tag{143}$$

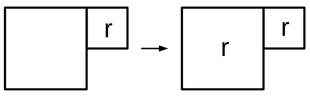
at all interior faces, where Δx_L and Δx_R are the cell sizes of the adjacent cells.

The flagging loop discussed above, gives us no such guarantee for the flag distribution it produced. Instead, we achieve this by traversing the interior faces again, in the grading loop. For each face we check the refinement level L and the preliminary flags in both the adjacent cells. If the combination of levels and flags would give a situation that violates (143), then we either remove a merge flag, or add a refine flag. Examples of this are shown in Figure 10. We limit the number of combinations to handle, by assuming that (143) holds when the grading loop starts. By induction, this is true if the initial mesh satisfies (143), and the grading loop is run for every adaptation.

If the adjacent cells are at equal levels $L_L = L_R$ there is only one illegal flag combination, which is shown in Figure 10a. The figure also shows how this is solved. If the cells are at different levels, there are multiple illegal flag combinations, listed in Table 4. This table also shows



(a) Merge flag is removed.



(b) Refine flag is added.

Figure 10: Examples of illegal flag and level combinations, and following countermeasures to avoid size ratio 4. *m* and *r* denote the merge and refine flags. The figures are taken from my project report [1], with small changes.

how these situations are solved. The first row might seem wrong, but as will be discussed in subsection 5.6, it is not guaranteed that a cell will merge if it has the merge flag at this stage. Figure 11 shows a detailed explanation of one of the illegal combinations in Table 4. The figure shows both why the combination is illegal, and how it is solved.

Every time a refine flag is added to solve an illegal combination, there is a chance that we create a new illegal combination. A simple, but costly solution for this would be looping over the interior faces multiple times, until no illegal flag combinations exist. Instead of doing that, each time we give a cell the refine flag to solve a combination, we add a pointer to this cell to a container. A queue structure [14, 18], which follows the first-in-first-out (FIFO) policy, was used in the present AMR solver. After we have looped over all the interior faces, we start looping over the cell pointers in the queue. For each of these cells, we check all the enclosing faces and look for the illegal combinations in Table 4 and Figure 10a. Each time a cell is given a refine flag, we still push a cell pointer to the queue. When the queue is empty, we are guaranteed that there are no illegal combinations left.

After grading is finished, we know that all the flags are set according to the refinement criterion, the mesh grading constraint, and the upper and lower limits for the refinement level L. The flags are then processed in two separate loops, discussed in the two following subsections.

5.5 Refining

The refine loop is a loop over all the cells in the mesh, where we check the refine flag r_i for each cell i. If $r_i = 1$ we split cell i into four new cells. This is done by inserting three copies of the parent cell i into the linked list, directly after the parent, as shown in Figure 12b. Thus, the parent cell becomes the first child cell, and the remaining 3 children cells come directly after it in the list. When the initial mesh is created, the cells are added from the lower left domain corner, row by row, left to right, as in Figure 12a. Using this strict cell ordering scheme for both the initial mesh and when refining, has no penalty when inserting and deleting in a linked list. However, it does allow us to find all the cell positions at any time using their refinement levels, as mentioned in subsection 5.1. This internal ordering of the cells does *not* mean that the mesh is structured. I.e. the ordering scheme does not specify connectivity between cells

Table 4: Flag combinations for cell pairs at different levels. The countermeasures are listed in the right column, if the combination is illegal. Lowest level means biggest cell and highest level means smallest cell. This table is taken from my project report [1] and given small changes.

Lowest level	Highest level	⇒ Reaction
□	▫	
m	m	Lowest looses m
m		Lowest looses m
m	r	Lowest gets r
	m	OK
		OK
	r	Lowest gets r
r	m	OK
r		OK
r	r	OK

and faces. It is used to relate children cells to each other while adapting, and to find cell centers *after* the actual simulation is complete.

Before the parent is copied, some of its data members are changed, to affect all the children cells:

$$r_i := 0 \quad , \quad V_i := \frac{V_i}{4} \quad , \quad L_i := L_i + 1 \tag{144}$$

The first assignment makes sure all cells have $r = 0$ the next time we start flagging. The second divides the cell volume by 4, and the third increments the refinement level.

Creating new objects by copying others is often referred to as copy-construction. After copy-constructing the 3 new cells, all the 4 child cells are identical, i.e. all their data members (see Table 3) are equal. For the flags r, m, $\tilde{m}$, the volume V, and the level L, this is exactly what we want. However, for the flow variables,

$$\rho, \; u, \; v, \; p, \; \rho u, \; \rho v, \; \rho E \quad , \tag{145}$$

and the exact, cell-evaluated fluxes

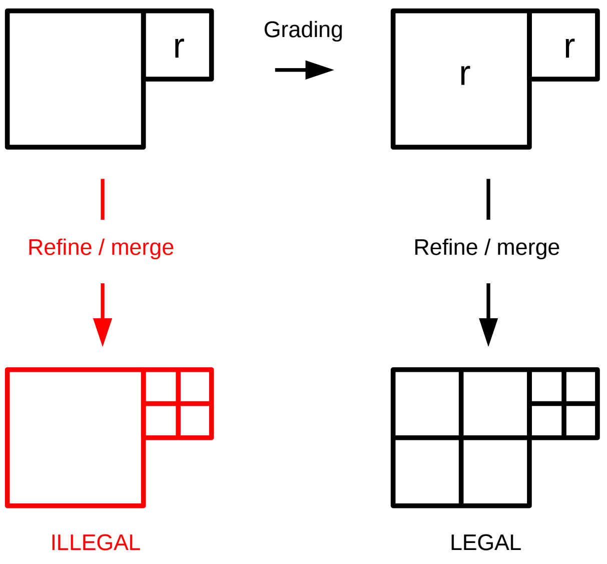
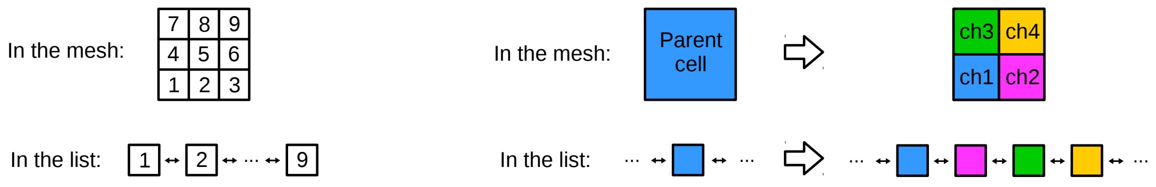


Figure 11: Example of an illegal flag combination, corresponding to row 6 in Table 4. The red part of the figure shows how the result would be if if the refine and merge procedures were run directly after flagging, without grading. In the red result, the largest cell is 4 times larger than the small cells, implying that they are 2 refinement levels apart. The black arrows represent the proper procedure. Adding a refine flag to the largest cell, gives the case on row 9 in Table 4, which is legal. The figure is from [1].



(a) Example of a 3x3 initial mesh. The numbers indicate the traversal order in the list.

(b) A cell is refined, shown by the large white arrow. The colors show that the parent cell becomes child 1, and how the other children cells are ordered.

Figure 12: The scheme for ordering cells in the linked list. The upper part in both subfigures shows how cells are positioned in the mesh. The lower parts show the cells' order in the linked list. The small black double-ended arrows represent pointers between list nodes. The dots $\cdots$ represent multiple cells that were not drawn.

$$F_{\rho u},\ F_{\rho v},\ F_{\rho E},\ G_{\rho v},\ G_{\rho E} \quad , \tag{146}$$

there are pros and cons. On the positive side, copying the flow variables ensures that the solution stays exactly the same as before the mesh adaptation. We will see examples of this in the results in subsections 6.1 and 6.3. On the negative side, the convergence indicator

$\|\Delta\rho^n\|_1 / \Delta t^n$ (130) will sky-rocket if the mesh is adapted while the solution is converging. We will examine this in the results as well, specifically in Figure 33. Interpolation of the flow variables using values in neighbor cells was considered, but not implemented.

The remaining data members are the iterators to the enclosing faces, providing the cell-to-face connectivity. Since these are copied from the parent, most of the iterators point to incorrect faces. In fact, several of the faces the they will point to, do not exist yet. Figure 13 shows why at least 4 new faces must be created for every refined cell. The figure also shows that we may have to split several of the enclosing faces, depending on the levels of the neighboring cells. Faces are split by dividing its area A_k by 2, and copy-constructing another face.

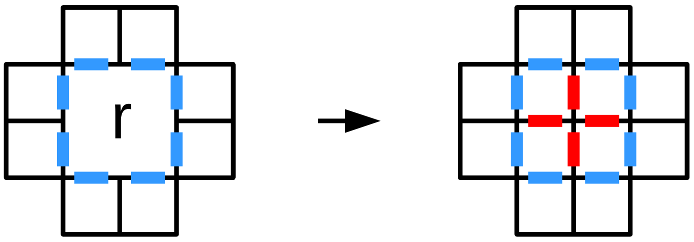
Newly created faces are inserted into the linked lists for interior vertical faces, interior horizontal faces, or one of the boundaries. For the interior faces we try to follow a certain ordering scheme, as with the cells. When we are looping over interior faces, we always access both the adjacent cells. I.e. flux transfer, flagging and grading. We want the next face in the loop to access one of the cells that we accessed from the current face in the loop, as seen in Figure 14. This was meant to utilize cache locality in time [19]. In the initial mesh we achieve this by creating the interior vertical faces row by row, left to right, similar to the cells in Figure 12a. Conversely, the interior horizontal faces are created column by column, bottom to top.

When refining cells, we try to sustain this order by building horizontal chains of vertical faces, as seen in Figure 15, and vertical chains of horizontal faces. In Figure 15a we see a parent cell that has two right-faces. They are number k and l in the list, and we do not know how many faces that come between them. To expand the potential chains that follow k and l, we insert the new faces before k and l, as shown in Figure 15a. The best-case in said figure is if $j = k - 1$ before refining (left of the arrow). If $j = k - 1$ before refining, then $j = k - 2$ after refining, which makes two unbroken chains: $\{l - 2,\ l - 1,\ l\}$ and $\{j,\ k - 1,\ k\}$. In Figure 15b, we use the same idea, but build from the left in case there are chains before j and k. Similarly to 15a, there is a best-case for 15b, if $l = k + 1$ before refining.

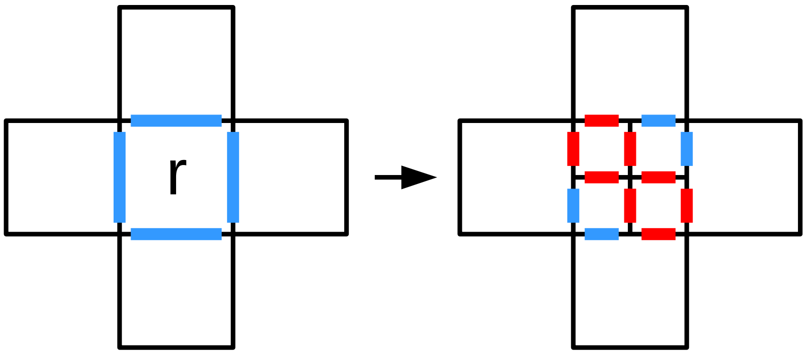
In Figure 15c there are no chains to expand, so we create two new ones. In the current implementation this is done by inserting all the new faces before k, as the figure suggests. In hindsight I supect it would be more cache-friendly to change $k - 4$ and $k - 3$ in 15c into $j + 1$ and $j + 2$. The latter approach would guarantee two unbroken chains of three faces each.

Following the schemes in Figure 15 promotes the cache hits discussed in Figure 14, but it will not sustain the order perfectly. E.g., in the right part of Figure 15d we see that face number k and face number $m - 1$ are adjacent to the same cell in the mesh, but they are not subsequent in the linked list. When the parent cell has two left-faces and two right-faces, as in Figure 15d, there can be chains at both sides. These chains cannot be spliced in the current implementation, but it could be achieved using move-semantics [18].

The same approaches are used for horizontal faces. To build and sustain vertical chains of



(a) Since all the surrounding cells were one level higher than the refined cell, none of the enclosing (blue) faces needed splitting. However, several of them needed to update connectivities. 4 new (red) faces were created, to separate the children cells. This extremely unlikely, but not impossible, situation gives the lowest possible number of new faces to create.



(b) Since all the surrounding cells were at the same level as the refined cell, all the enclosing faces (blue) were split. We can see that the blue faces were not deleted, but their areas are changed. 8 new (red) faces were created. As always, 4 of them separate the children cells. The other 4 come from splitting blue faces.

Figure 13: Examples of cell refinement and its consequences for the faces. In both cases the middle cell is refined while the surrounding cells are unchanged. Blue faces existed before the refinement, but some of their data members may have changed during the refinement. Red faces were created by the refinement and inserted into the linked lists for faces. This figure is taken from [1] and adapted slightly.

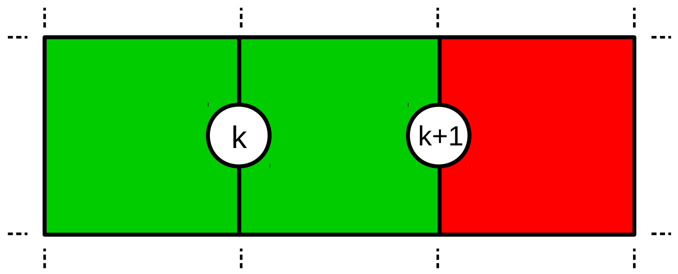


Figure 14: Three contiguous cells in a mesh. The white circles on two of the faces show their order in the list of interior vertical faces, where k is directly before $k+1$. Face k is the current face in the loop, and is accessing its adjacent (green) cells. The next face $k+1$ is likely to get a cache-hit when accessing its left adjacent (green) cell [19]. If face $k+1$ had been above, below, or far away from k, we would not get this cache hit.

horizontal faces, the schemes in Figure 15 can be mirrored over the rising diagonal. This symmetry can be seen by examining figures 13b and 15c in combination.

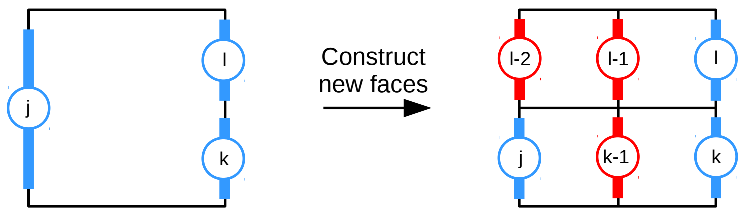
The algorithm for creating new faces is summarized in the following numbered list:

1. Check if we can expand a chain.
2. Create the faces between the children cells, following one of the schemes in Figure 15.
3. Split enclosing faces if necessary. If they are interior, follow the appropriate scheme in Figure 15. If an enclosing face is at a domain boundary, split it and put the new face in the appropriate boundary linked list.
4. Where necessary, update face areas, and set connectivities (pointers and iterators), such that all faces point to the correct cells, and vice versa.

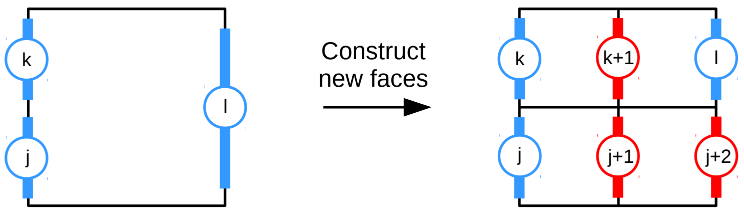
These 4 steps are done first for vertical faces, then for horizontal faces, separately. Locating the appropriate faces is very simple, using the cell-to-face connectivity, k_{w1} , k_{w2} , k_{n1} , k_{n2} , k_{e1} , k_{e2} , k_{s1} and k_{s2}.

In addition to checking flags and refining cells, we execute several other operations in the refining loop:

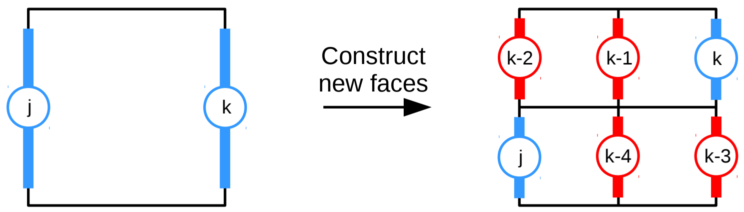
- We find the highest occurring refinement level $\max_i\{L_i\}$, and use it to find $\min_i\{\Delta x_i\}$ in the end of the loop. This is necessary to respect the CFL condition (111).
- We set the prevent merge flag to $\tilde{m}_i = 0$ for all cells.



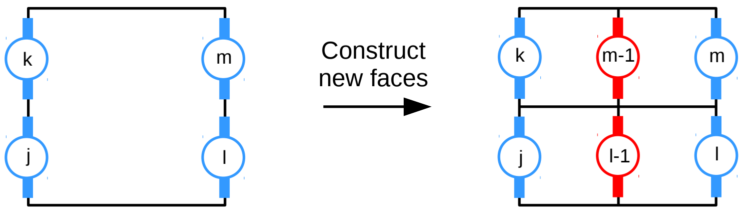
(a) Parent had 2 right-faces, so we build from the right.



(b) Parent had 2 left-faces, so we build from the left.



(c) Parent had 1 right-face and 1 left-face, so we start new chains.



(d) Parent had 2 right-faces and 2 left-faces. We build from the right, but cannot splice with the chains on the left.

Figure 15: Building horizontal chains of interior vertical faces, when cells are refined. Colored lines with circles are interior vertical faces. Red faces are created when following the arrow, blue faces are re-used. The indices j, k, l and m inside the circles denote the traversal order in the list of interior vertical faces. $k+1$ is after k; and $k-1$ is before k; etc.

- We update the indicators' minimum and maximum values, which gives the largest spreads (138). They are needed for computing the refine and merge thresholds.

- We store an iterator to the former parent cell, which is now the first child cell (see Figure 12). By keeping iterators to all "child 1" cells, we simplify the merging loop considerably, as we will discuss in subsection 5.6.

5.6 Merging

The merging loop is the final stage of each mesh adaptation. Cells are merged in groups of four, as illustrated by Figure 1 in subsection 1.1, and the four cells must stem from the same cell refinement. To ensure this, we utilize the ordering of the cells in their linked list, shown in Figure 12. If we know that cell i was child 1 in a cell refinement (ch1 in Figure 12), and the subsequent cells i, $i+1$, $i+2$ and $i+3$ are at the same refinement level, then the four cells came from the same refinement. We loop over the cell iterators that point to ch1 cells, which are stored whenever cells are refined. For each cell i we overwrite the merge flag m_i such that

$$\begin{aligned} m_i :=& \\ & m_i \ \& \ m_{i+1} \ \& \ m_{i+2} \ \& \ m_{i+3} \ \& \\ & L_{i+1} = L_i \ \& \ L_{i+2} = L_i \ \& \ L_{i+3} = L_i \end{aligned} \tag{147}$$

In (147) we demand that the cells i, $i+1$, $i+2$ and $i+3$ originated from one single cell refinement, and that they all have the merge flag. Thus, if a cell is flagged for merging after the flagging and grading loops, it is not guaranteed that the cell can merge. If $m_i = 1$ after applying (147), we merge the four cells. This reverses the refinement that created them, that is, $i+1$, $i+2$ and $i+3$ are deleted and i is enlarged to fill their void. Before we delete any cells, we set

$$L_i := L_i - 1 \quad , \quad V_i := 4\,V_i \quad , \quad m_i := 0 \quad . \tag{148}$$

I.e., the refinement level and volume of the remaining cell are corrected, and we remove its merge flag. The last part is important because multiple iterators in this loop can point to the same cell.

The flow variables and cell-evaluated fluxes (see "floating points" in Table 3) are set in the remaining cell i, to the arithmetic mean of the four merging cells, e.g.,

$$\mathbf{U}_i^n := \frac{\mathbf{U}_i^n + \mathbf{U}_{i+1}^n + \mathbf{U}_{i+2}^n + \mathbf{U}_{i+3}^n}{4} \quad . \tag{149}$$

This changes the solution, but the total mass in conserved. After all of the flow variables are set as in (149), the four faces that separated the children cells are deleted. In Figure 13a, four such faces are colored red. The four merging cells are enclosed by exactly 8 faces, as we can see by comparing the right parts of figures 13a and 13b. We may need to assemble several of those. As opposed to the splitting of faces in subsection 5.5, two faces are assembled by deleting one and doubling the area of the other. To find out which faces to assemble, we check the following for the two faces at each side:

- Are the faces at the boundary?
    - If so, assemble them.
    - If not, check the level(s) L_{nb} of the neighbor cell(s) at that side, and compare with the level L_i of the four merging cells.
        * If $L_{nb} = L_i$, do not assemble.
        * If not, then we know that $L_{nb} = L_i - 1$. $\Rightarrow$ Assemble the faces.

We find and delete faces using the iterators k_{w1} , k_{w2} , k_{n1} , k_{n2} , k_{e1} , k_{e2} , k_{s1} and k_{s2} from the merging cells. After the appropriate faces are deleted or assembled, we correct the connectivities. That is, we update the cell pointers in the faces and the face iterators in the cells, such that they reflect how cells and faces are connected. Finally, we delete the three cells $i+1$, $i+2$ and $i+3$; and we delete the cell iterator that pointed us to cell i.

6 Results and Discussion

In this section, the obtained results will be displayed and discussed. Before examining the results, we will see *how* the data was measured and presented.

All the 1D plots in this report are prepared with the Matplotlib library in Python. This is a free open source multi-platform library, specialized in plotting.

All the 2D plots that show solutions of flow variables, are prepared by a cell plotting tool that I have made myself, but not specifically for this project. The plotting tool was made using the free open source multimedia library SFML. This cell plotter represents data as colored rectangles, where the color of each rectangle is constant. This resembles how the approximated cell averages (107) in the numerical solutions are constant throughout each cell. In a solution from the AMR-solver, the rectangles' sizes are different, showing the cell sizes. This can be accentuated by outlining all the rectangles, where the outlines will show the faces that enclose the cells. In a solution by the structured solver all the rectangles are the same size.

All the measurements are done by the solvers themselves. When the total CPU runtime τ is measured, some of the procedures are excluded: Finding the analytical solution, computing the numerical errors, storing data for the cell plotter, and saving a report file, are not included in the total runtime. Some minor book-keeping to produce average cell count, convergence history, etc. are included, but the complexities of these operations are low enough to neglect them. What is *not* negligible, is the fact that the solvers run through an operating system (Linux Mint 18.3, 4.15.0-50-generic). Even though no other tasks were given the computer while simulating, hidden processes can interrupt the CPU and give a measurement uncertainity for the runtime. This uncertainity has not been tested thoroughly, but running identical simulations 3-4 times typically gives a relative spread of 1-5 %.

6.1 Development of a Shock Reflection

The shock reflection test case was introduced in subsection 3.1. In this subsection we will examine how the shock reflection flow develops, from the initial condition to the steady-state solution. The main reasons for studying this are:

1. To check that the mesh adapts as intended. We can check if the fine resolution areas follow the shocks.

2. To provide intuitive understanding of the flow.

3. To check that the solution actually converges, and achieves a steady state.

Table 5: Parameters and derived parameters used by the AMR-solver to simulate the development of the shock reflection.

(a) Given parameters.

Δx_{max} :	$\frac{1}{10}$
L_{max} :	4
η_r :	0.04
η_m :	0.0182

(b) Available refinement levels and corresponding cell sizes.

					L_{max}
L :	0	1	2	3	4
Δx_{amr} :	$\frac{1}{10}$	$\frac{1}{20}$	$\frac{1}{40}$	$\frac{1}{80}$	$\frac{1}{160}$
	Δx_{max}				Δx_{min}

We will primarily study the results from a simulation by the AMR-solver, with the parameters listed in Table 5. The allowed refinement levels and cell sizes are listed in Table 5b. These are driven parameters, dictated by Δx_{max} and L_{max}. For later simulations, only Δx_{max}, L_{max} and Δx_{min} will be listed. All the results displayed in subsection 6.1 will be from the AMR simulation with the parameters in Table 5, unless something else is specified.

Figures 16 - 21 are cell plots of the density ρ at different time instants. The time instants, shown in the tops of the figures, were handpicked to show the most interesting features. The colorbars on the right side of the figures show the symbol and value of the plotted variable. These colorbars are scaled to the entire series, and are therefore identical in figures 16 - 21. The plots are prepared with the cell plotter, introduced at the beginning of section 6.

In Figure 16 we can recognize the initial conditions (36) discussed in subsection 3.1. We see that the mesh around the discontinuity has already been refined to the highest allowed level $L_{max} = 4$, before the first time step is computed. Since physical variables are copied from parent cells to children cells, we can see that the initial refinement did not change the solution.

In Figure 17 we see that the shock has propagated downwards, while retaining its intersection with the y-axis at y_α. We also see that the shock has lost the sharpness that we saw in the initial conditions. Since there is no diffusion introduced by the Euler equations (22), this must be numerical diffusion. The mesh in Figure 17 shows that many cells have been refined since the start, keeping the shock well resolved. However, almost no cells have been merged, leaving a wide trail of high resolution above the shock. This is not a bug, it is a feature. The refine criterion monitors density ρ, but also the velocity components u and v. We shall later see that a gradient in u is the main reason for this refined area.

In Figure 18 we see that the shock has moved further down and that it is still well resolved. We also see that the highly refined area above the shock is becoming separated from the rest of the high resolution cells.

In Figure 19 the shock has started to interact with the wall at the lower boundary. The density then increases rapidly close to the wall, between the reflection point and the right boundary.

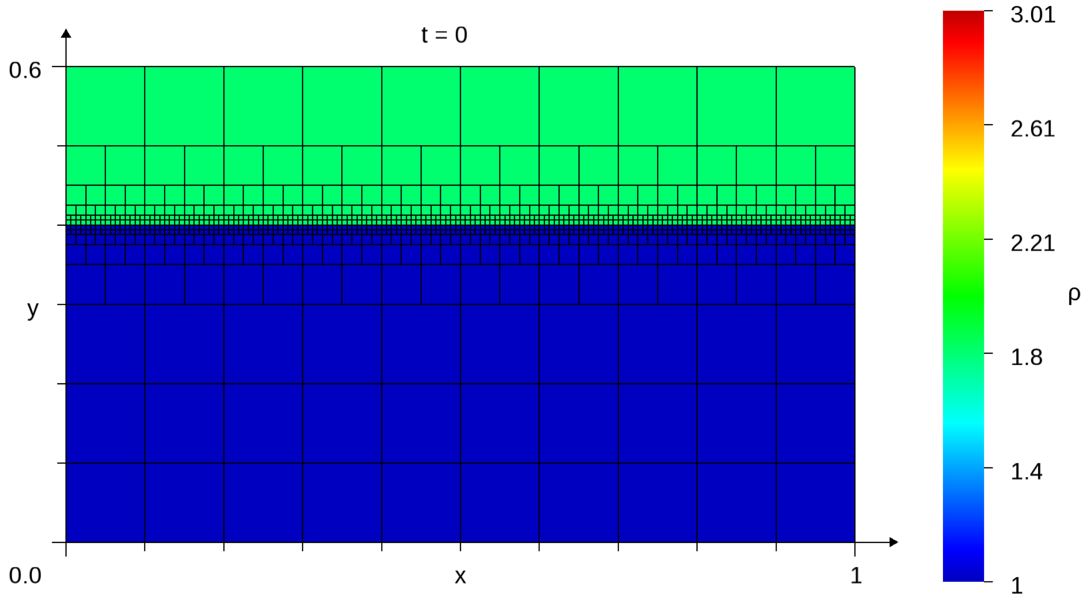


Figure 16: Cell plot of density initial condition.

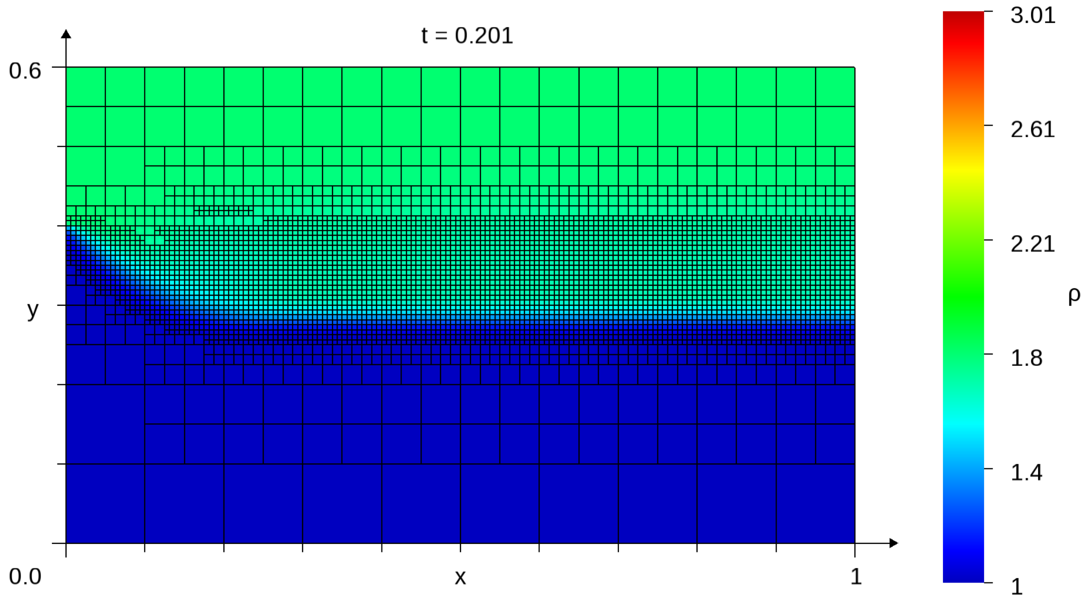


Figure 17: Cell plot of density at $t = 0.201$. The shock propagates downward. A large refined area forms above the shock.

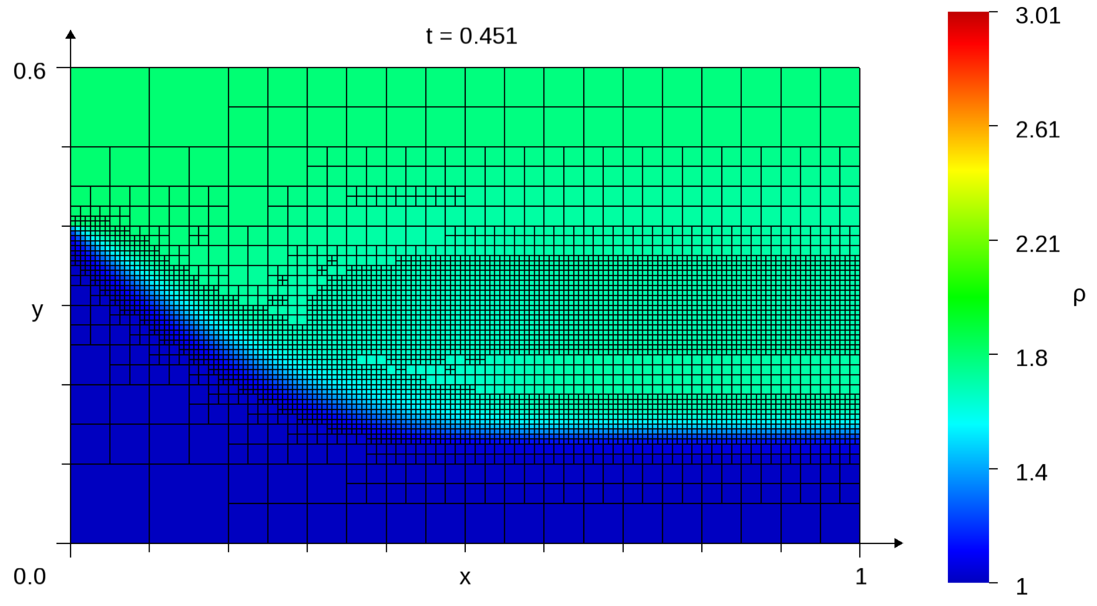


Figure 18: Cell plot of density at $t = 0.451$. The refined area above the shock gets clearly distinct from the density discontinuity.

This is the formation of region 3, which was discussed in subsection 3.1. The shock and the emerging reflection is well resolved by small cells, and the refined area above the shock has detached from the area around the shock.

In the time between Figures 19 and 20, the incoming shock has straightened, which has caused the reflection point to move leftward. Figure 20 also shows that the reflected shock is bending upwards, increasing the angle β from Figure 2. At this stage, the mesh seems well adapted to the density field. We have high resolution around the shock, and coarse cells almost everywhere else.

The reflected shock keeps bending upward until the solution becomes steady, as it converges. The converged solution is shown in Figure 21, which indicates that the mesh is still very well adapted to the incoming and reflected shocks.

The x-component u of the velocity develops slightly different from the density. Figures 22-27 show this development. Again the colorbars are scaled to the entire series of u-plots, causing the colorbars to be identical in Figures 22-27. The time instants in these figures were handpicked, some to compare with the density plots

Figure 22 shows the initial condition for u. Similar to the density, the initial u discontinuity moves downwards and is subjected to numerical diffusion, as seen in Figure 23. However, we also see that the right and horizontal part of the wavefront, for $x \gtrsim 0.4$, seems to propagate slower than the left part. An extra plot at the same time instant is given in Figure 24. The simulation that produced Figure 24 used a higher base resolution $\Delta x_{max} = \frac{1}{20}$, higher $L_{max} = 5$

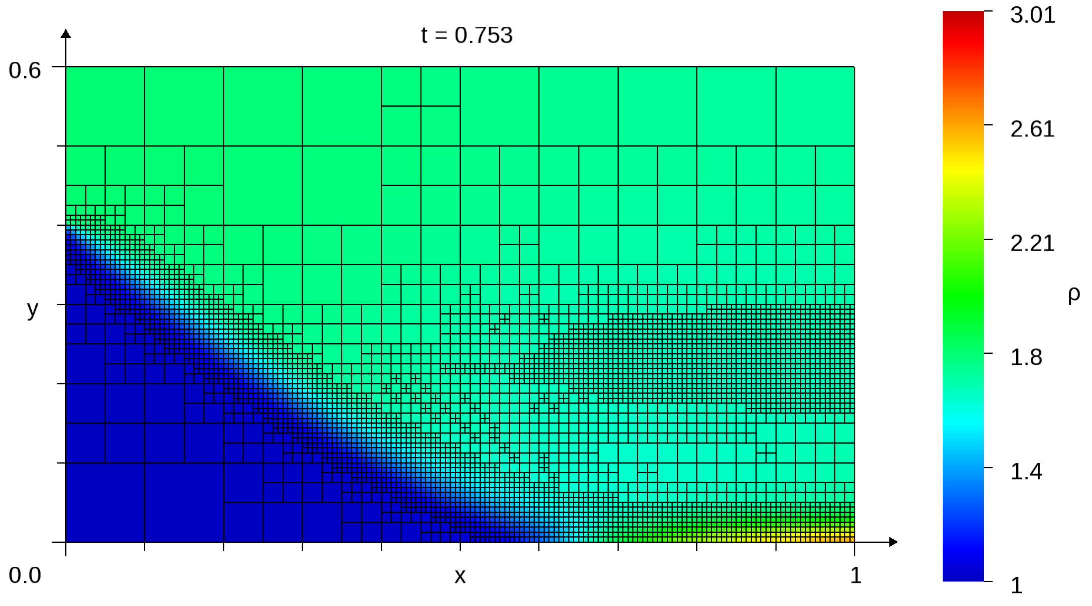


Figure 19: Cell plot of density at $t = 0.753$. The shock hits the wall and starts to reflect.

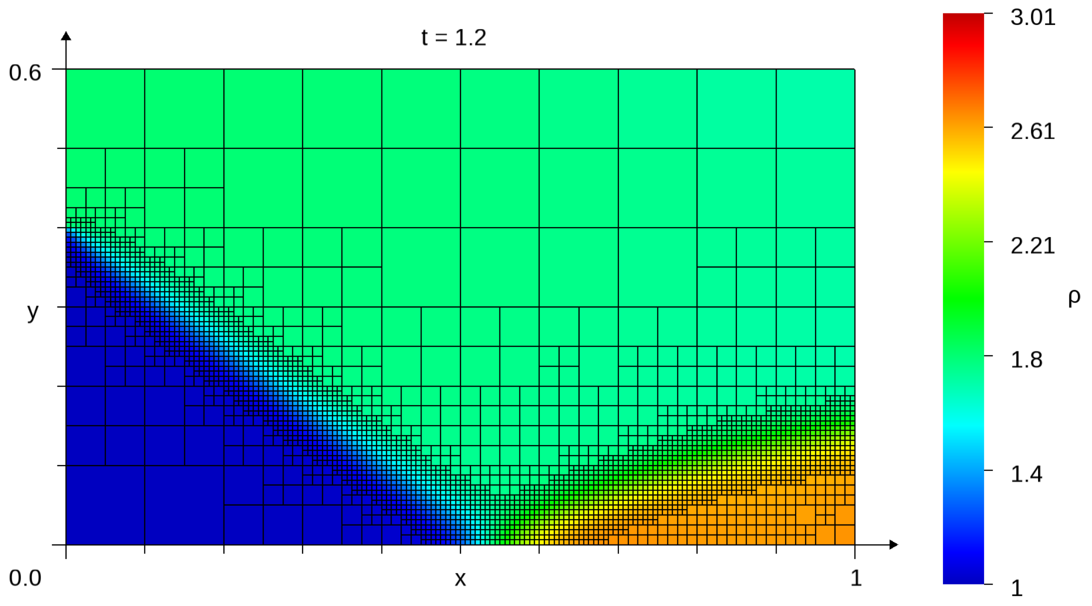


Figure 20: Cell plot of density at $t = 1.2$. The reflection point moves leftwards and the reflected shock bends upward.

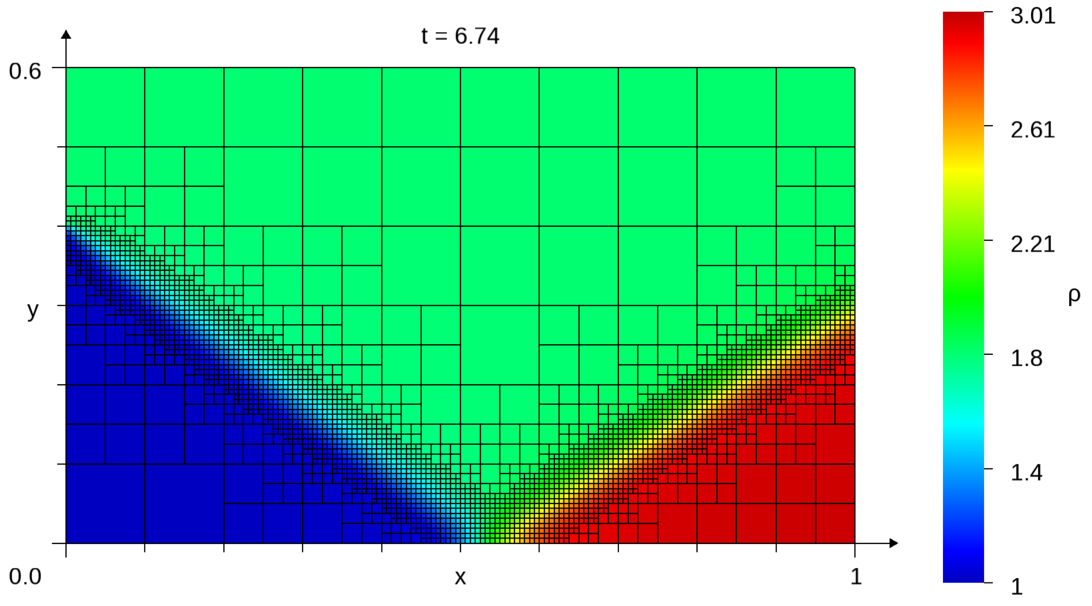


Figure 21: Cell plot of density at $t = 6.74$. Steady state is achieved.

and a stricter refine tolerance. The consequence of adjusting the parameters like this is that Figure 24 has much smaller cells than the other plots, and thus resolves the gradients much better. Figure 24 reinforces the impression that the wavefront is split into two parts, and that the right part propagates slower. It also shows an S-shaped transition area between the left and right parts. Now, comparing Figures 18 and 23 shows why the fork-shaped area of refined cells is necessary, to resolve both ρ and u.

In Figure 25 the S-shaped transition area has become so diffuse that most of the cells in that area have merged. What is left of the right wavefront is then advected out of the domain through the right boundary, and the shock hits the wall as seen in Figure 26. The reflected shock then forms extensionally like a telescope, while bending upwards at the same time. At this stage we do not observe highly refined cells in any other area than around the shocks. This suggests that all the variables that are monitored by the refinement criterion have coinciding gradients. When the reflected shock finds it steady position, the solution converges. The steady state solution is shown in Figure 27.

Now we examine the velocity component v, parallell with the y-axis. Figures 28-32 show its development. From the initial condition shown in Figure 28, the wavefront propagates very similarly to the density field. We can observe this by comparing Figures 29 and 18.

When the shock reaches the lower boundary, as seen in Figure 30, the lower part of the shock becomes aligned with the wall. It then starts to bend upwards, creating the reflected shock as seen in Figure 31. The solution then proceeds to its steady state and converges as shown in Figure 32. Figures 30-32 also indicate that v is well resolved throughout the flow development.

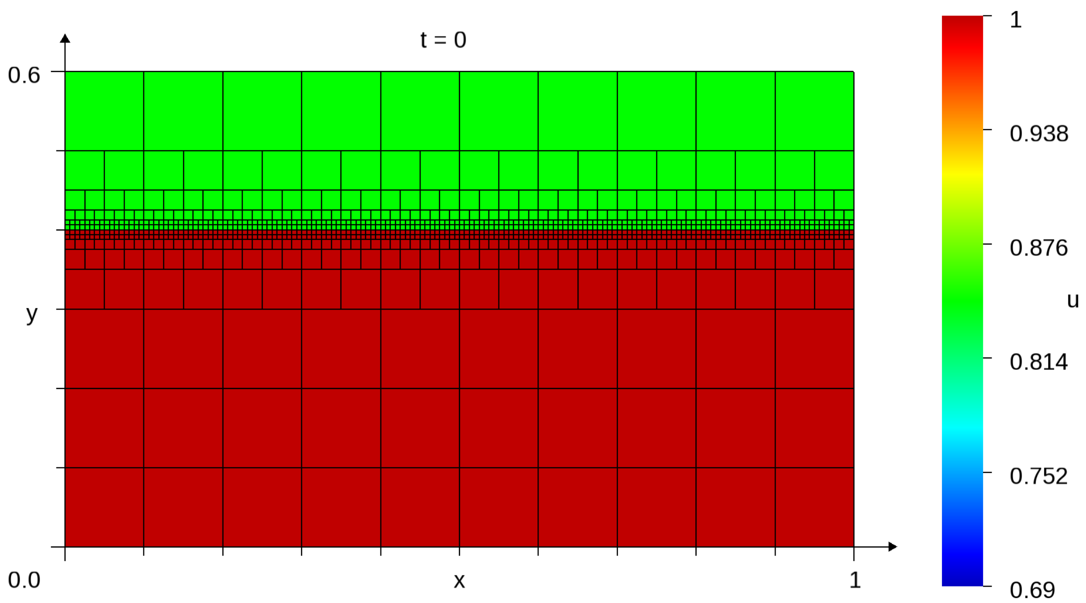


Figure 22: Cell plot of the initial condition for velocity component *u*.

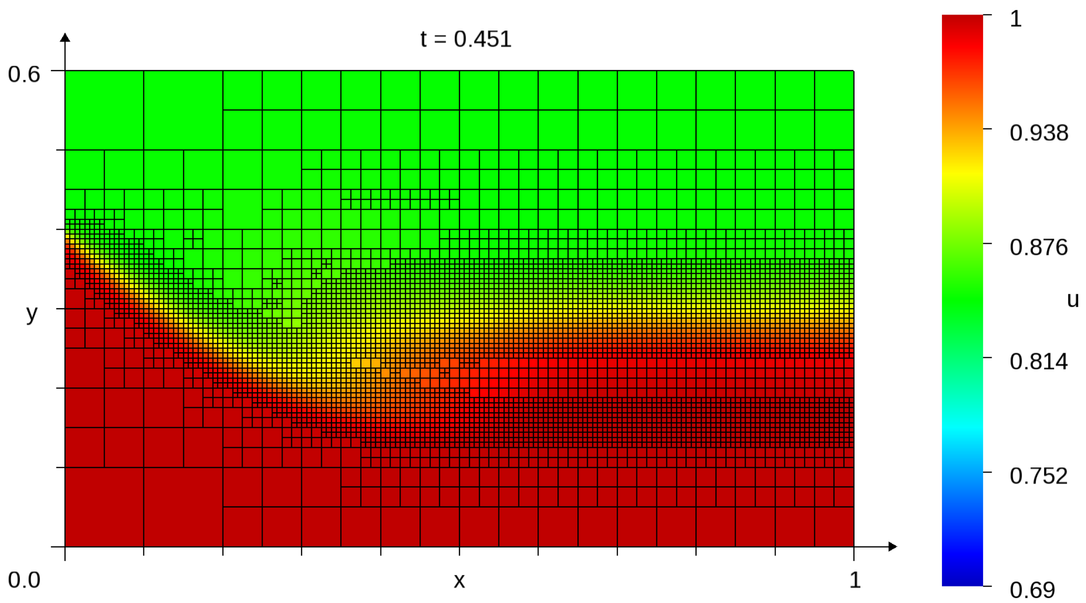


Figure 23: Cell plot of velocity component u at $t = 0.451$. The shock propagates downwards. $\Delta x_{max} = \frac{1}{10}$ and $\Delta x_{min} = \frac{1}{160}$.

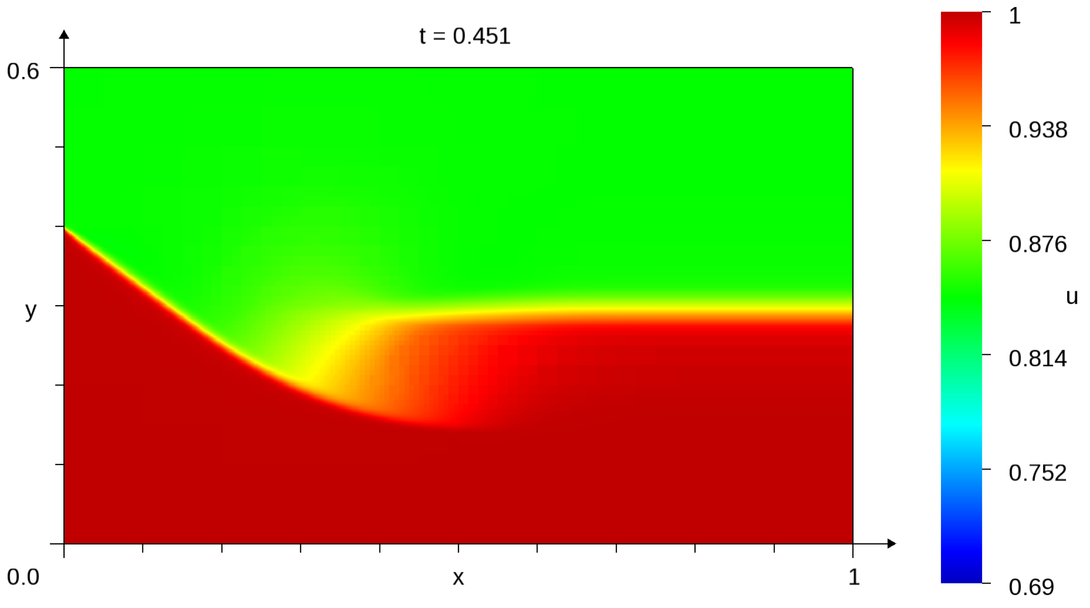


Figure 24: Cell plot of velocity component u at the same time instant $t = 0.451$ as Figure 23, but with $\Delta x_{max} = \frac{1}{20}$ and $\Delta x_{min} = \frac{1}{640}$, and without drawing faces.

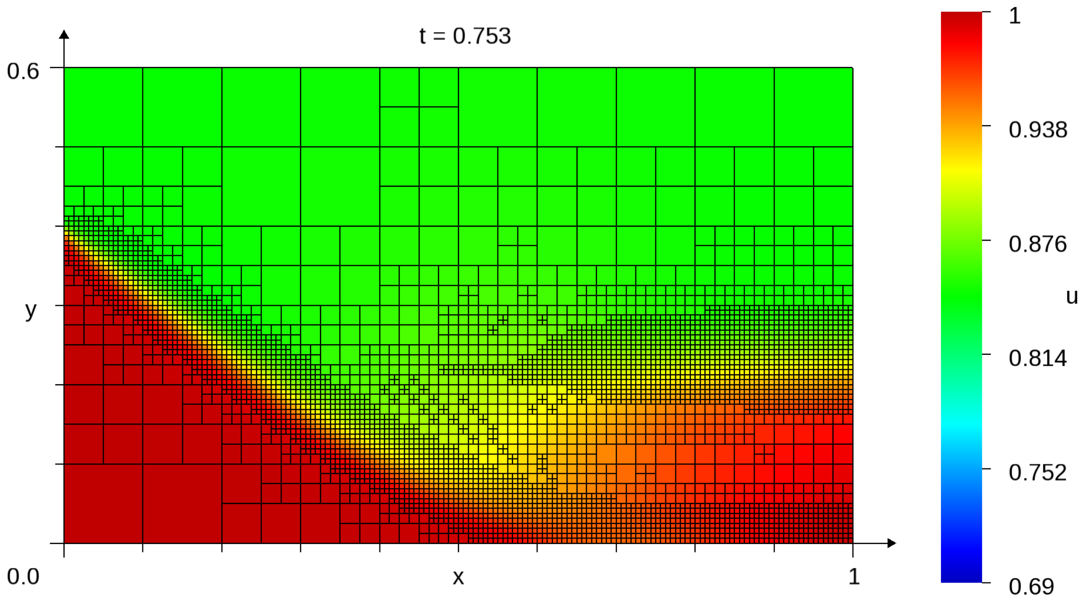


Figure 25: Cell plot of velocity component u at $t = 0.753$, just before the shock hits the wall and starts to reflect.

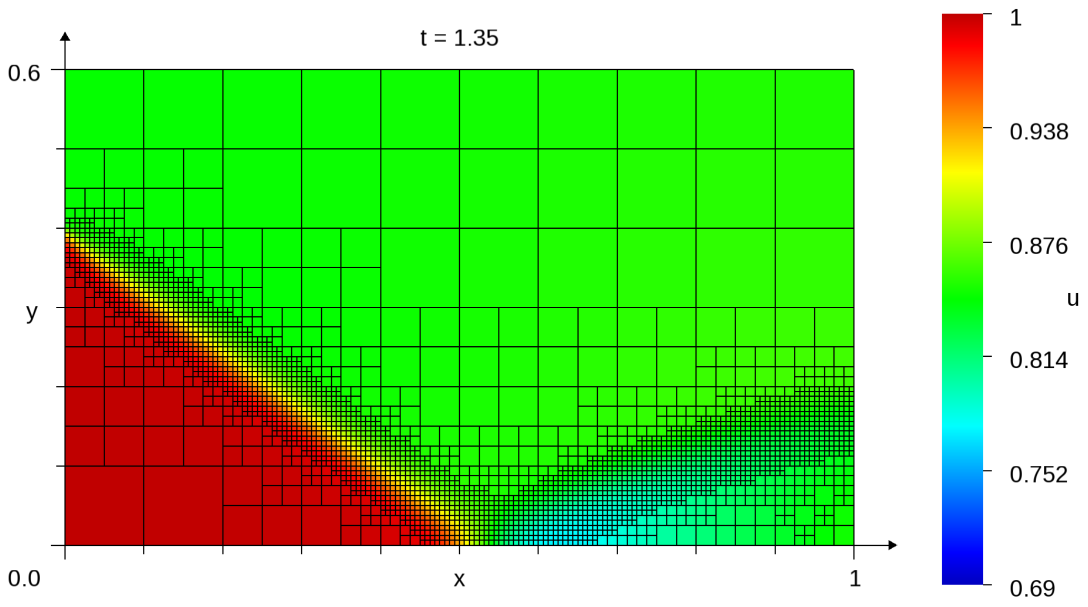


Figure 26: Cell plot of velocity component u at $t = 1.35$. The reflected shock forms extensionally, while bending upwards.

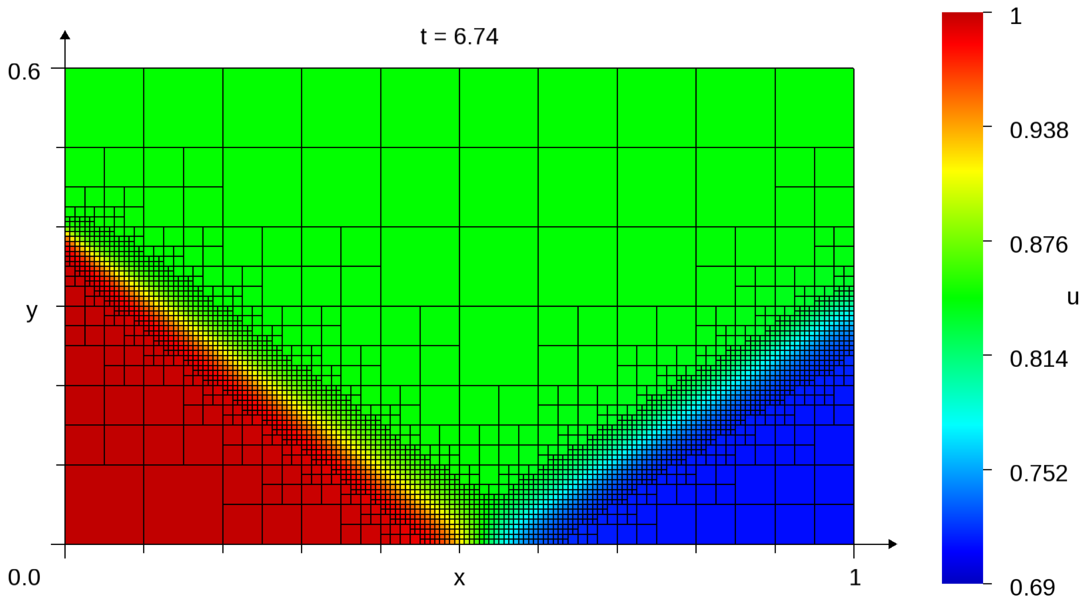


Figure 27: Cell plot of velocity component u at $t = 6.74$. Steady state is achieved.

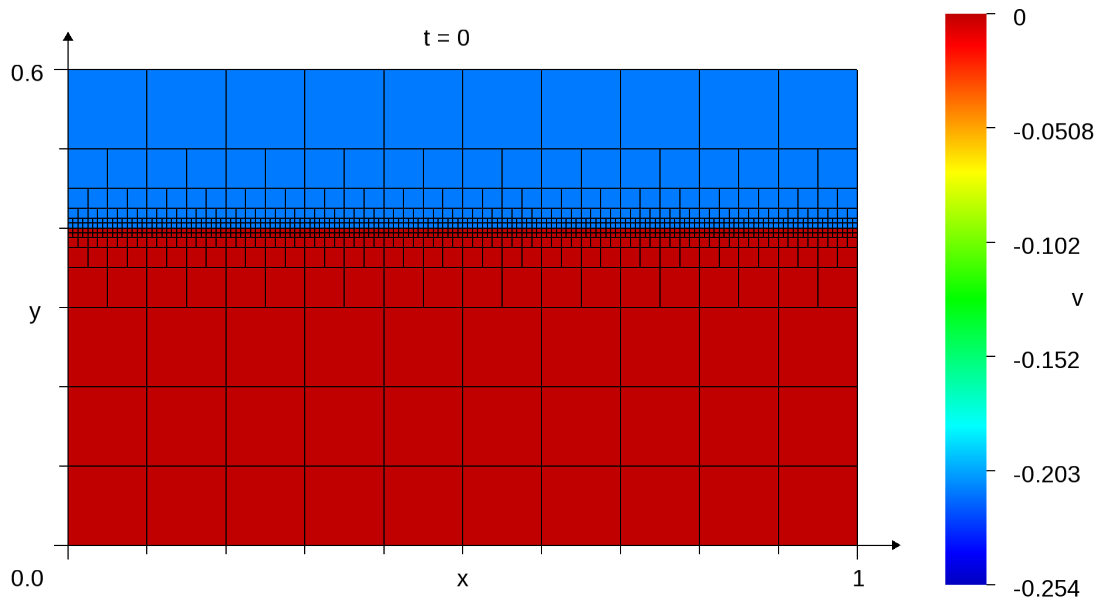


Figure 28: Cell plot of the initial condition for velocity component v.

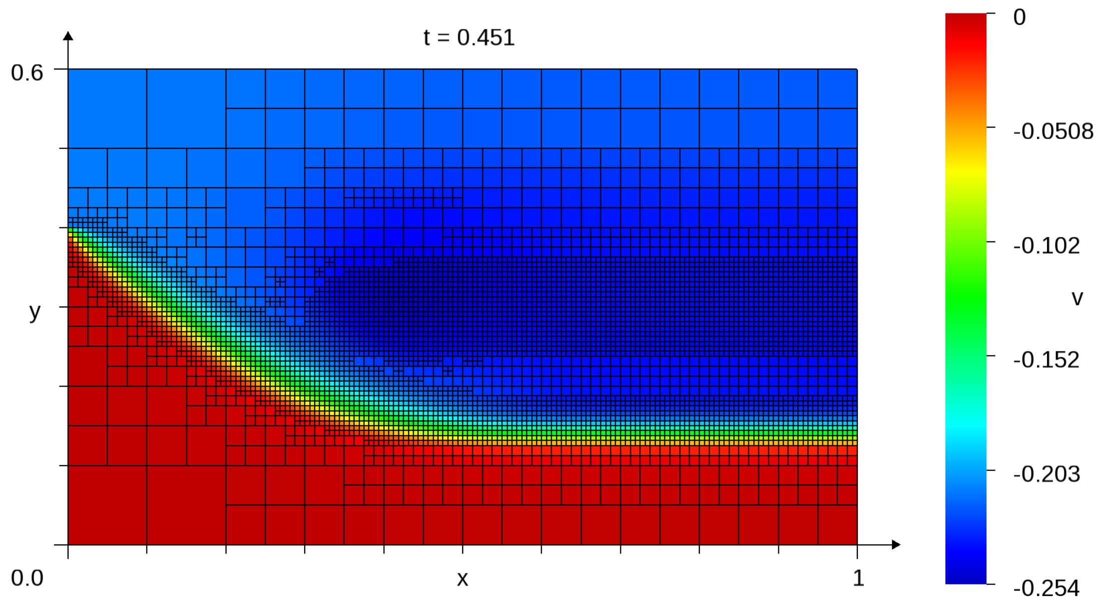


Figure 29: Cell plot of velocity component v at $t = 0.451$. The shock propagates downwards, and almost exits the refined area.

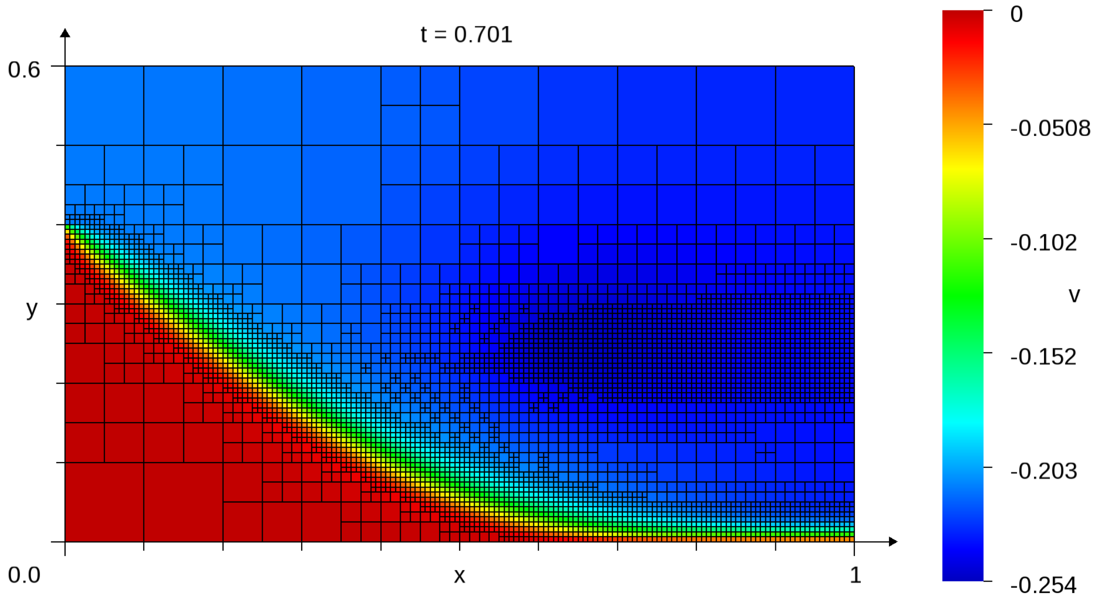


Figure 30: Cell plot of velocity component v at $t = 0.701$. The shock hits the wall and starts to reflect.

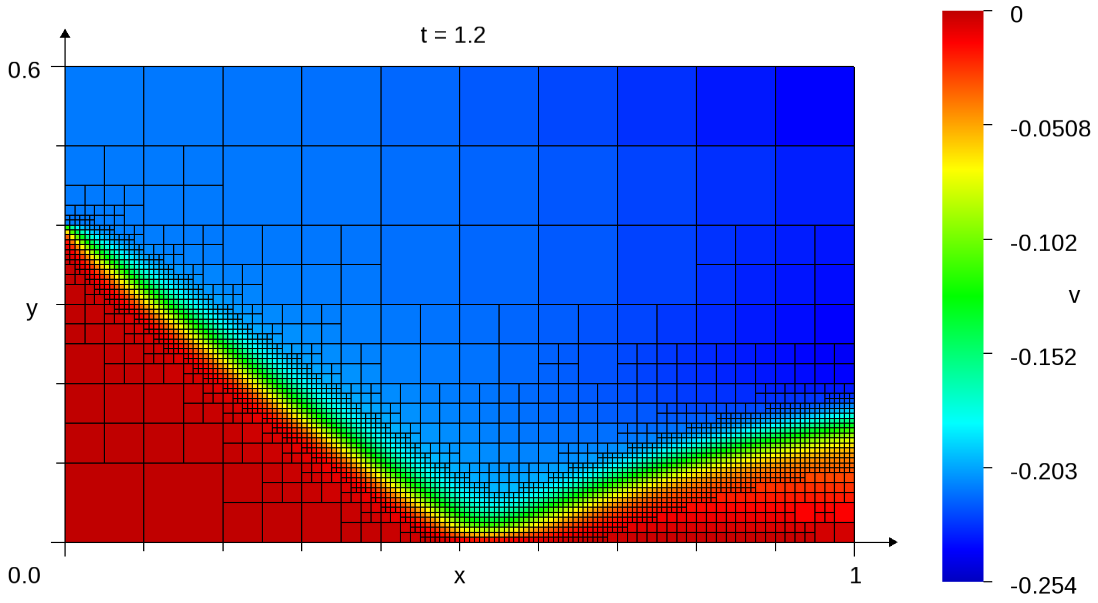


Figure 31: Cell plot of velocity component v at $t = 1.2$. The reflected shock bends upwards, while the reflection point moves to the left.

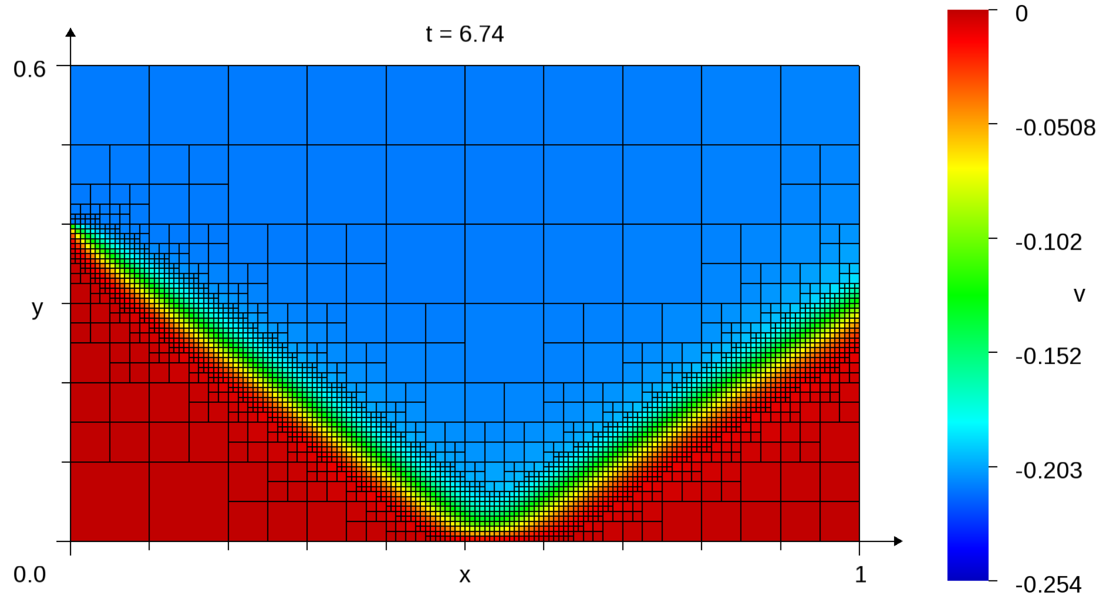


Figure 32: Cell plot of velocity component v at $t = 6.74$. Steady state is achieved.

The pressure p develops very similarly to the densityρ. The only visible difference is the values at the colorbar.

To check that the solutions discussed above achieve steady states, the convergence history of the AMR simulation cf. Table 5 is plotted in Figure 33. Another simulation was executed, by the standard finite volume solver solver (see subsection 4.7). It used a grid spacing $\Delta x_{str} = \Delta x_{min}$, i.e., equal to the smallest cell size allowed in the AMR-simulation. This standard simulation is also included in the convergence history plot in Figure 33. In said figure, the red graph shows that the convergence indicator $\|\Delta\rho^n\|_1 / \Delta t^n$, scaled by its initial value, decreases monotonically for the structured solver, until reaching the threshold, 10^{-4}. For the AMR simulation the convergence indicator is indicated by the blue graph in Figure 33. It peaks at the time levels indicated by the dashed vertical lines, because the mesh is adapted at those times, causing disturbances of the numerical solution. In this particular case, the convergence threshold is reached at $n = 2248$ for the AMR solver, and at $n = 1671$ for the standard solver. This means that the AMR solver takes 1.35 times as many time steps as the structured solver, which is a recurring trend. Although the factor varies between 1.3 and 1.6, the AMR-solver required more time steps to converge in all simulations.

It seems natural that the mesh adaptation disturbs the convergence. Figure 33 confirms this, by showing that the last peaks in the blue graph coincide with the dashed vertical lines. However, the figure also shows that the blue graph displays a strong linear tendency, for $n \gtrsim 500$ in this logarithmic plot. At the last few mesh adaptations we can see that the graph recovers very quickly back to its linear trend, and the peaks do not seem to affect this trend in the long run. The red graph in Figure 33 has no peaks, and more importantly, it displays a negative curvature

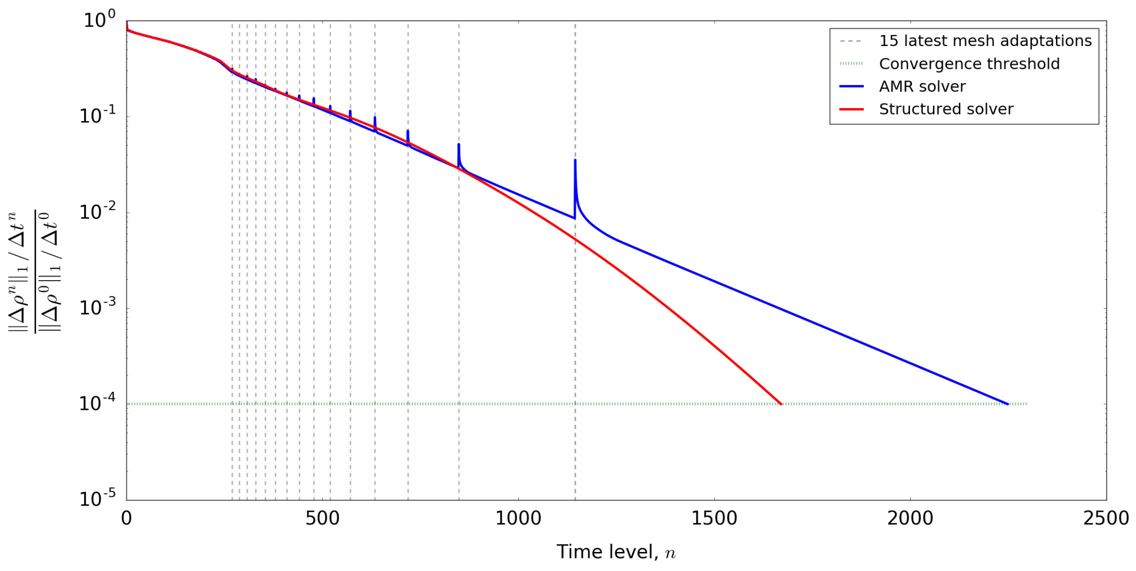


Figure 33: Convergence history plot with logarithmic ordinate axis. The red and blue graphs show the development of the convergence indicator $\|\Delta\rho^n\|_1 / \Delta t^n$, normalized by its initial value. The dashed gray vertical lines show the time levels when the last 15 of the total 42 adaptations occurred. Both solvers used $\Delta x_{min} = \Delta y_{min} = \frac{1}{160}$.

for $n \gtrsim 500$. Even though this difference in convergence rate is interesting and important to note, investigating it further is beyond the scope of this report.

Figure 33 illustrates yet another important point: If we observe the spacing between the dashed gray lines, it is evident that the mesh adaptations happen much more seldom later in the simulation. This relates to the adaptation trigger that was introduced in subsection 5.2. As $\|\Delta\rho^n\|_1 / \Delta t^n$ decreases, more time steps are required before mesh adaptation is triggered.

6.2 Measuring the Benefit from AMR

We will measure the gains from the AMR approach by comparing several AMR simulations with several structured simulations. The gain here refers to a reduction in the number of cells N_i, and a reduction of the CPU runtime τ. The reduction in τ shows the overall gain from this AMR approach with the current implementation. It factors in overhead in the data structure and algorithm efficiency. The reduction in cell count N_i is important because it shows the potential of the approach. It gives a measure of how much more we can decrease τ by optimizing the data structure and the algorithms.

In addition to measuring the gains we will measure the drawback of the approach, which is an increase in numerical error. We need to examine the relation between gains and drawbacks to see the actual benefit from the AMR approach. The error considered here is the density error

Table 6: Parameters that vary within an AMR simulation series. Δx_{max} is the size of all the cells when the mesh is created and L_{max} is the highest permitted refinement level. Δx_{min} is the smallest permitted cell size, and follows directly from Δx_{max} and L_{max}.

Sim. number:	1	2	3	4	5	6	7	8	9	10	11
Δx_{max} :	$\frac{1}{10}$	$\frac{1}{15}$	$\frac{1}{10}$	$\frac{1}{15}$	$\frac{1}{10}$	$\frac{1}{15}$	$\frac{1}{10}$	$\frac{1}{15}$	$\frac{1}{10}$	$\frac{1}{15}$	$\frac{1}{20}$
L_{max} :	2	2	3	3	4	4	5	5	6	6	6
Δx_{min} :	$\frac{1}{40}$	$\frac{1}{60}$	$\frac{1}{80}$	$\frac{1}{120}$	$\frac{1}{160}$	$\frac{1}{240}$	$\frac{1}{320}$	$\frac{1}{480}$	$\frac{1}{640}$	$\frac{1}{960}$	$\frac{1}{1280}$

ε_ρ (131), which was defined in subsection 4.6. The errors of all the primitive variables (25) and the Mach number were logged and examined, and they all show the same trends. Therefore, only results of ε_ρ are presented.

Instead of comparing simulations one to one we will present the data in plots and look for general trends. The simulations are grouped into 6 series, where 1 series was executed by the structured solver and the remaining 5 series were simulated by the AMR solver. The purpose of these simulations is to look for relations $\tau(\varepsilon_\rho)$ and $N_i(\varepsilon_\rho)$. We will compare these relations for each of the 6 simulation series graphically. Therefore, we must ensure that ε_ρ falls into a similar range for all the series. The error depends strongly on the smallest allowed cell size $\Delta x_{min} = \Delta y_{min}$ in the AMR-solver, and on the uniform cell size $\Delta x_{str} = \Delta y_{str}$ in the structured solver. This was utilized when setting the simulation parameters. In each of the 5 AMR series, the parameters that control the available cell sizes are varied according to Table 6. To exemplify: The first simulation in all the AMR series will allow cell sizes down to $\Delta x_{min} = \frac{1}{40}$, the second simulation in all the AMR series allows cell sizes down to $\Delta x_{min} = \frac{1}{60}$, and so on.

The 5 AMR series use different values for the refine and merge tolerances, η_r and η_m, which were introduced in subsection 5.3. The values are given in Table 7, were the merge tolerances have been set to

$$\eta_m = \frac{\eta_r}{2.2} \quad . \tag{150}$$

The first series has high tolerances. This gives relaxed refine and merge criteria, meaning that less cells will be refined, and more cells will merge. The last series has low tolerances. This gives stricter criteria, forcing more cells to be refined and permitting fewer cells to merge.

The structured series consist of 13 simulations executed by the structured solver, using equidistant grids. The 13 simulations use different mesh resolutions listed as grid spacings in Table 8. These chosen cell sizes are similar to Δx_{min} in Table 6, which was intended to give errors in the same range.

We start by examining the gains and drawbacks separately by plotting $N_i(\Delta x_{min})$, $\tau(\Delta x_{min})$

Table 7: Refine and merge tolerances for the 5 AMR series. η_r and η_m are defined in subsection 5.3.

Series no.:	η_r	η_m
1	0.08	0.0364
2	0.06	0.0273
3	0.04	0.0182
4	0.02	0.0091
5	0.01	0.0045

Table 8: Cell sizes, i.e. grid spacings for the structured solver.

Sim. number:	1	2	3	4	5	6	7	8	9	10	11	12	13
Δx :	$\frac{1}{35}$	$\frac{1}{40}$	$\frac{1}{60}$	$\frac{1}{80}$	$\frac{1}{120}$	$\frac{1}{160}$	$\frac{1}{240}$	$\frac{1}{320}$	$\frac{1}{480}$	$\frac{1}{640}$	$\frac{1}{960}$	$\frac{1}{1100}$	$\frac{1}{1200}$

and $\varepsilon_\rho(\Delta x_{min})$. Figure 34 shows that the AMR simulations use much fewer cells than the structured solutions in the left part of the figure, where we allow small cells. We also see that the refine tolerance has a large impact on the cell count. Series 1 has the most relaxed tolerance, $\eta_r = 0.08$. Its cell count is almost two orders of magnitude below the structured cell count in the left end of the graphs. Overall, the cell count increases when the tolerances decrease, except for the left end of the green graph, which is series 3. This simulation was run again multiple times, but gave the same strange values every time. It is included in all the plots and it stands out in almost all of them, but will not be investigated further in this report.

Next we plot the CPU time $\tau(\Delta x_{min})$ in Figure 35. From this plot we see that the runtimes are considerably lower for the AMR simulations than for the structured ones. In the left end of the plot the AMR simulations have smaller cells than the structured solver, but use only 10 - 35 % of the runtime. I.e., we see the same trend as for the cell count, but to a smaller degree. The refine and merge tolerances seem to have an even bigger effect on runtime than on cell count. With the most relaxed tolerances (red), the highest resolved simulation (leftmost) is one order of magnitude below the structured solver. With the strictest criterion (blue), the ratio between AMR and structured is close to 3.

The error $\varepsilon_\rho(\Delta x_{min})$ is plotted in Figure 36. We can see from this plot, that the AMR simulations give higher errors than the structured ones. Especially the simulations with the high refine tolerance $\eta_r = 0.08$. Conversely, the simulations in series 5 (blue) with the strictest tolerances give errors that are quite close to the structured simulations, but they are still a little higher. From the slopes of the error plots, it does not look like the error

$$\lim_{\Delta x \to 0} \varepsilon_\rho = 0 \quad , \tag{151}$$

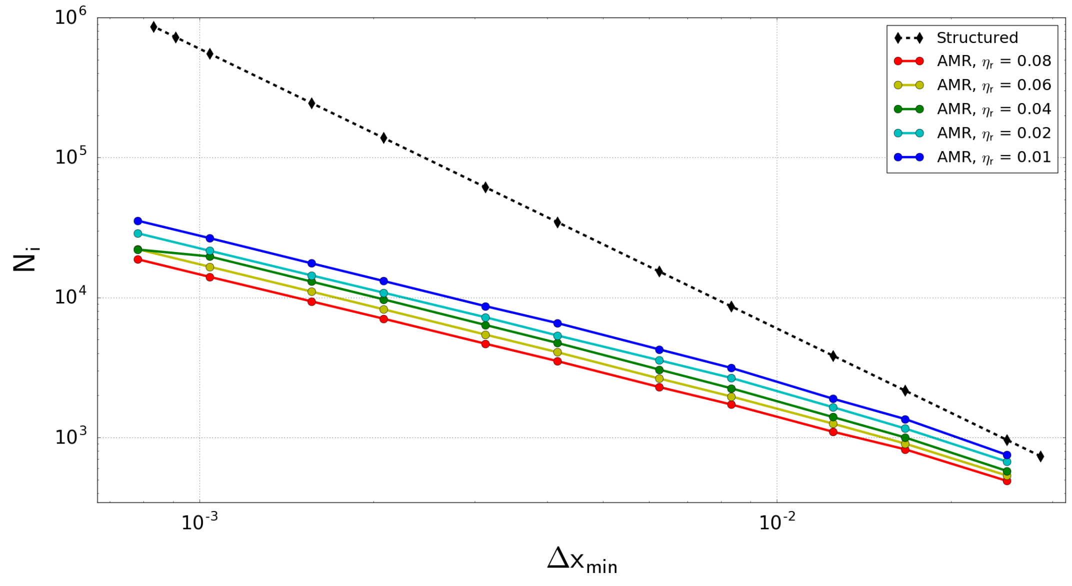


Figure 34: The final number of cells in the mesh, plotted against the smallest cell size Δx_{min}. For the structured solver, this is the size of alle the cells in the mesh. η_m is not shown in the legend since it is $\frac{\eta_r}{2.2}$ in all simulations. This applies to all following plots. Both axes use the logarithmic scale.

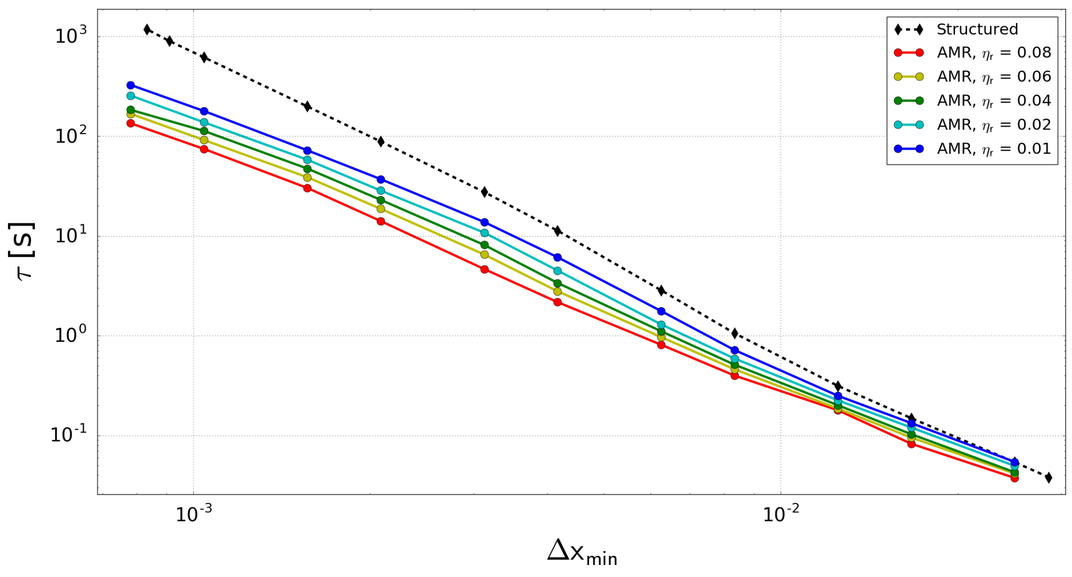


Figure 35: The simulation runtime (CPU time), plotted against the smallest cell size Δx_{min}. Both axes use the logarithmic scale.

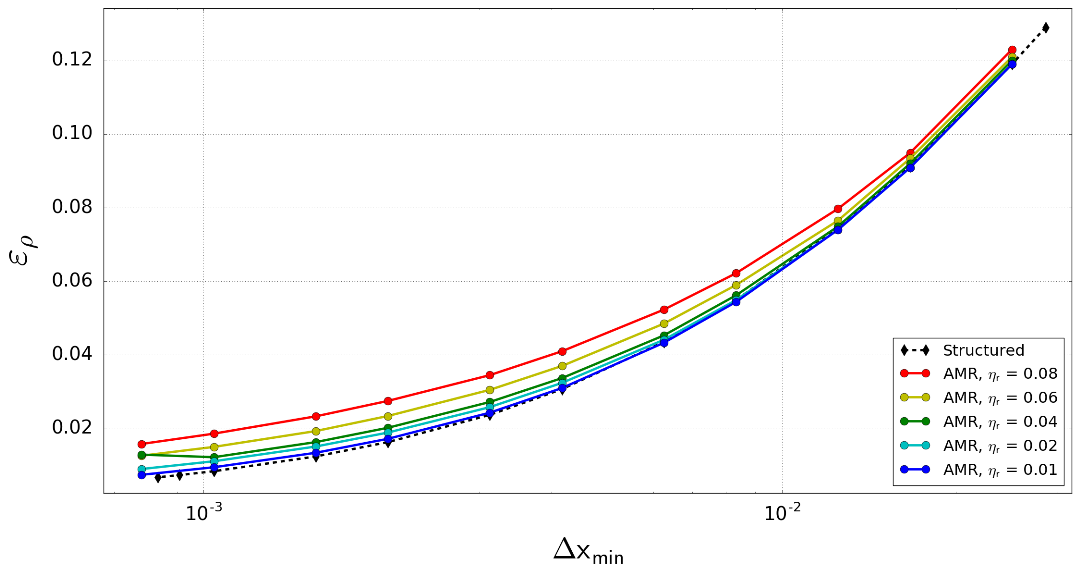


Figure 36: The density error ε_ρ (131), as defined in subsection 4.6, plotted against the smallest cell size Δx_{min}. The first axis uses logarithmic scale.

Table 9: Comparison between an AMR simulation and a standard simulation, where we see a large benefit from AMR. For the standard solver the refine tolerance η_r is not applicable, and Δx_{min} was the size of *all* the cells. N_i is the final number of cells in the mesh, τ is the total CPU time in seconds and ε_ρ is the density error. The ratios are the values from the standard solver divided by the AMR values.

	η_r	Δx_{min}	N_i	τ [s]	ε_ρ
AMR:	0.01	$\frac{1}{1280}$	35 433	327	0.00742
Standard:		$\frac{1}{1100}$	726 000	903	0.00735
Ratio:			20.5	2.76	0.991

for any of the solvers. This could be further investigated by allowing even smaller cells.

The trade-offs between gains and drawbacks are examined by plotting $N_i(\varepsilon_\rho)$ and $\tau(\varepsilon_\rho)$. In Figure 37 we see that all the AMR simulations use fewer cells than the standard solver to achieve a certain error. The simulations in the left end of the plot show the largest benefits, using only 5-10% of the number of cells. Details about one of these AMR simulations is listed in Table 9, where it is compared to a structured simulation that gave an approximately equal error. Figure 37 also shows a trend regarding the refine and merge tolerances. In the right half of the plot the AMR benefit increases with increasing tolerances. Conversely, at the left end of the plot the lowest tolerances give larger benefits. This indicates that larger tolerances are more beneficial for larger cells, and smaller tolerances are more beneficial for smaller cells, since the error decreases with Δx_{min}, cf. Figure 36.

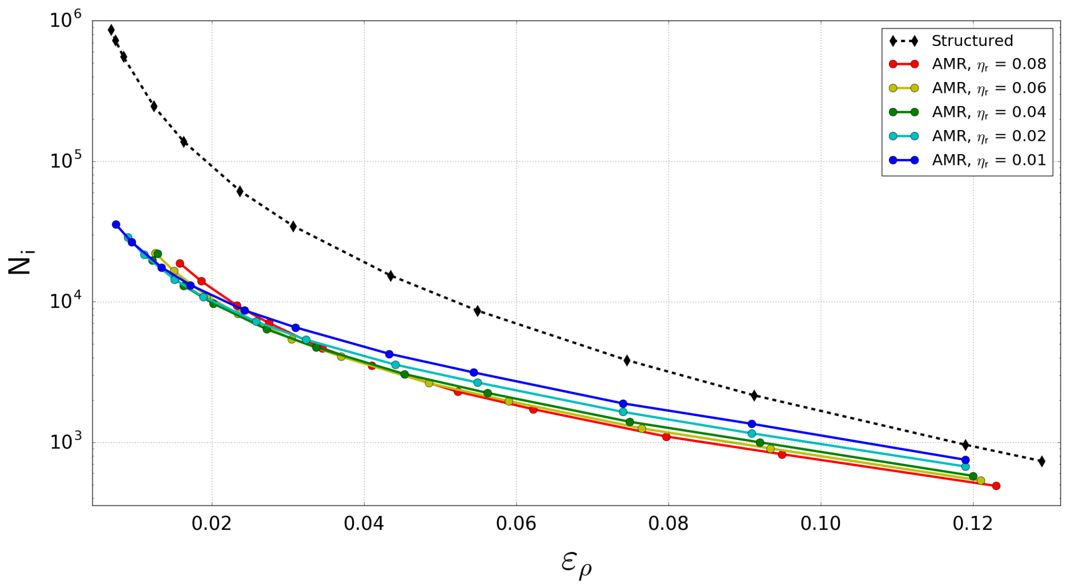


Figure 37: The final number of cells in the mesh, plotted versus the density error. The ordinate axis uses the logarithmic scale.

The plots of $\tau(\varepsilon_\rho)$ do not provide good comparisons with any of the four combinations of proportional and logarithmic axes. On proportional axes, the graphs of $\tau(\varepsilon_\rho)$ resemble rectangular hyperbolas with asymptotes at the coordinate axes. However, the product $\tau\,\varepsilon_\rho$ increases hyperlinearly as ε_ρ decreases, and the graphs appear too close to each other to distiguish. To provide a readable plot the runtimes were normalized as

$$\tau^* = \frac{\tau}{\tau_{str}} \quad , \tag{152}$$

where τ is the runtime the simulation in focus, and τ_{str} is the runtime of a simulation from the structured solver, that gives the same error as the simulation in focus. Since no two simulations have equal errors, τ_{str} is found by linear interpolation between structured simulations:

$$\tau_{str} = \tau^{right} + (\varepsilon_\rho - \varepsilon_\rho^{right})\frac{\tau^{left} - \tau^{right}}{\varepsilon_\rho^{left} - \varepsilon_\rho^{right}} \quad , \tag{153}$$

where the superscripts ^{left} and ^{right} denote the closest structured solution on each side, in terms of error. In Figure 38 the normalized runtime τ^* is plotted versus ε_ρ. The plot shows that the AMR simulations mostly took less CPU time per error than the structured simulations. However, the red graph shows that if the tolerances are too relaxed, we will not achieve this benefit in the left part of the plot. Even though the high tolerance $\eta_r = 0.08$ gives low runtimes and cell counts, it increases the error so much, that the standard solver could achieve the same error using less runtime. Like for the cell count N_i in Figure 37, we find the highest benefits in the left end of Figure 38. These AMR simulations use less than 40% of the CPU time that the

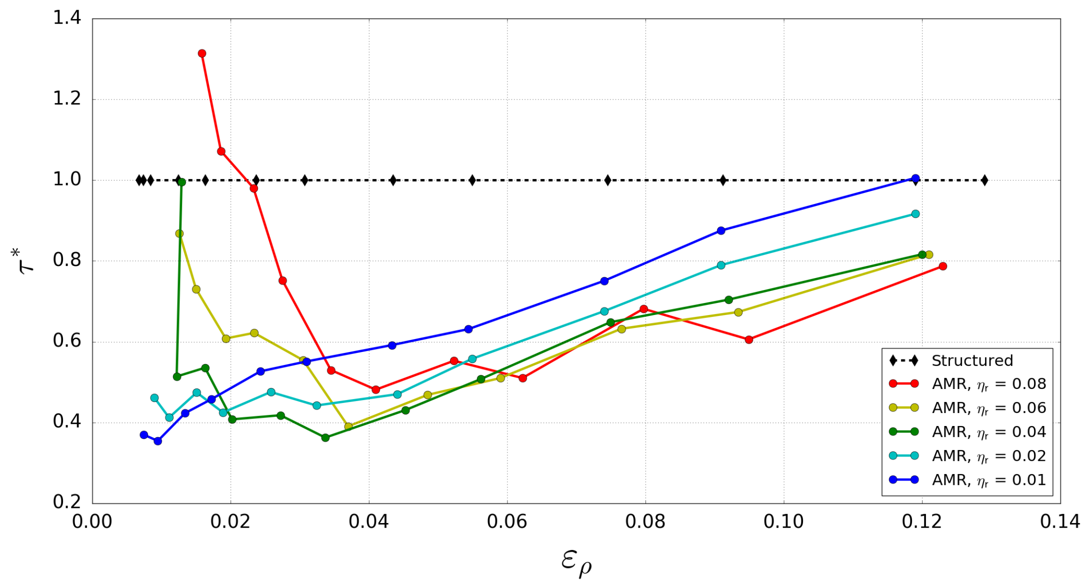


Figure 38: Normalized CPU runtime τ^* (152). This is plotted versus the density error ε_ρ.

standard solver uses to reach the same error. One of these AMR simulations is the one that was detailed in Table 9. For the refine and merge tolerance we see the same trend in figures 37 and 38: Smaller tolerances are more beneficial for smaller cells and larger tolerances are more beneficial for larger cells. If this relationship holds, the refine and merge tolerances should be set depending on the smallest allowed cell size.

6.3 Simulation of a 2D Riemann Problem

The second test case that was studied is a two-dimensional (2D) Riemann problem. It was introduced and discussed in subsection 3.2. For this flow case we do not have an analytical solution to verify the solution. However, there are other gains from simulating this test case:

1. We will check if the AMR-solver finds the same solution as the structured solver.
2. We will check if the mesh adaptation algorithm detects high gradients in a more complicated flow.
3. We will still compare runtimes between the AMR-solver and the structured solver.

The parameter settings that were used to simulate this test case with the AMR-solver are given in Table 10. The refine and merge tolerances η_r and η_m, and the smallest cell size $\Delta x_{min} = \Delta y_{min}$ are set relatively small, if compared to the parameters used in subsection 6.2. This was done to ensure that all the different wave types mentioned in subsection 3.2, were

Table 10: Parameters used by the AMR-solver to simulate the 2D Riemann problem.

Δx_{max} :	$\frac{1}{20}$
L_{max} :	6
Δx_{min} :	$\frac{1}{1280}$
η_r :	0.015
η_m :	0.0068

resolved. For the structured solver, the uniform grid spacing $\Delta x_{str} = \Delta y_{str}$ was set equal to the smallest allowed cell size for the AMR-solver.

As mentioned, we limit this test case to the time before any of the waves reach the domain boundaries. By trial and error it was found that the time interval

$$0 \leq t \leq 0.2 \qquad (154)$$

achieves this while still showing the waves we are interested in.

Figures 39-44 show the simulation results from the 2D Riemann problem test case. The time instant is shown at the top, and the size and name of the plotted variable is shown by the colorbar.

We start by confirming that Figure 39 agrees with the initial condition (99) given in subsection 3.2. We recognize the regions, with increasing density at higher x- and y-values. The IC for pressure is identical to density, and the ICs for the velocity components are constant zero. Their plots are therefore not included.

Figure 40a shows the final density solution at $t = 0$, computed by the AMR-solver. In this density field we can identify all the different waves, by using Figure 7 in subsection 3.2 as reference. We see the vertical shock at $x \approx 0.23$ and the horizontal shock at $y \approx 0.2$. The contact discontinuities at $x \approx 0.48$ and $y \approx 0.43$ are not as distinct, especially near the center of the domain. At $x \approx 0.7$ and $y \approx 0.7$ we find the rarefaction waves. They are wider than the other waves, as indicated in Figure 7.

We also recognize the different areas that were discussed in the end of subsection 3.2. I.e. we see that the density is constant in the corners, and equal to the initial condition. We also see the 1D Riemann problems along the artificial domain boundaries, where all the wavefronts are linear and perpendicular to the adjacent boundary. In the central area we observe the 2D effects, as non-linear wavefronts.

Figure 40b is a plot of the density at the same time and flow case as 40a, but computed by the structured solver. Comparing the two plots in Figure 40 gives a strong indication that the

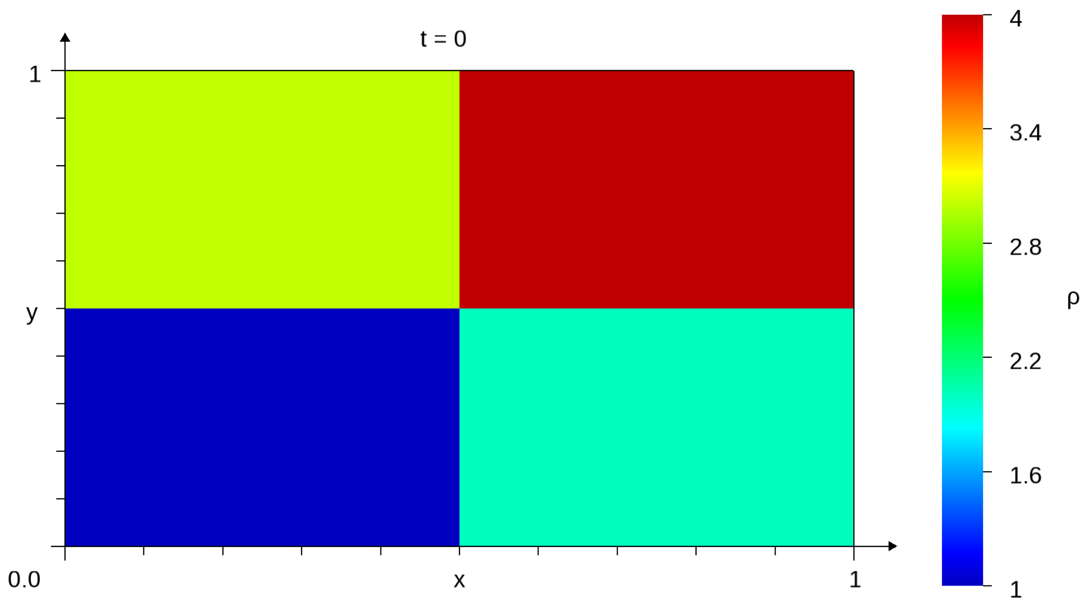


Figure 39: Initial condition for the density in the 2D Riemann problem.

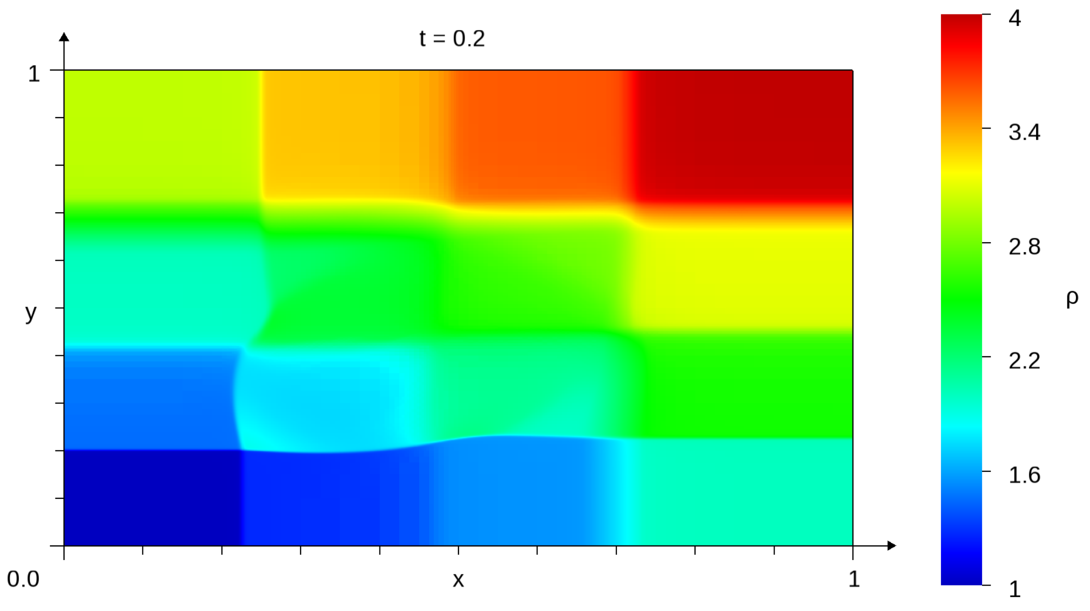
solvers have found the same solution. We can observe small differences around the contact discontinuities, where it is possible to make out the color changes between cell interfaces in the AMR solution. This indicates that the mesh might be too coarse in these areas.

Figures 41 and 42 are plots of the velocity components u and v, from the AMR solution. One thing we notice in these figures is that not all the waves are observable in these plots. The u-plot Figure 41 shows the entire vertical shock- and rarefaction waves, but not the vertical contact discontinuity. The horizontal shock and contact can be seen, but only between the outer vertical waves. Instead of a distinct horizontal rarefaction wave, we see that u increases gradually from the horizontal contact, up to $y \approx 0.7$, where the density showed the rarefaction wave in Figure 40.

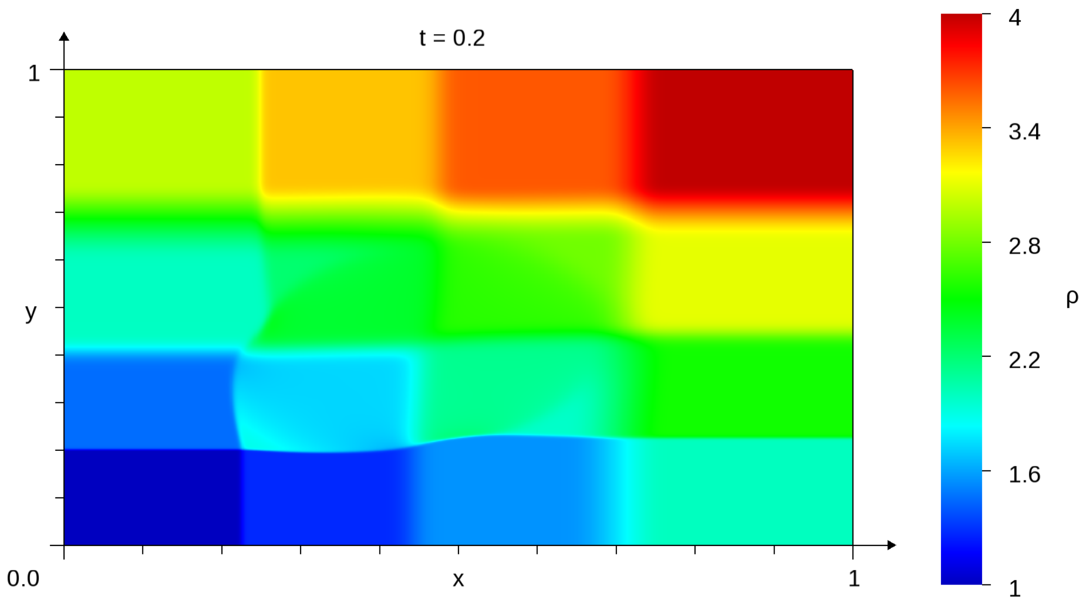
All the same features appear for the velocity component v in Figure 42, but mirrored over the rising diagonal. We can also compare these velocity fields to the blue arrows in Figure 7 and find consistency for the flow directions along the boundaries.

Comparing the AMR velocity fields in figures 41 and 42 with the result from the structured solver did not show any differences apart from those discussed when commenting the density. Velocity fields from the structured solver are therefore excluded.

The pressure from the AMR-solver's solution is plotted in Figure 43. It shows both the shocks and both rarefaction waves, although the vertical rarefaction wave is a little vague. None of the contact discontinuities can be observed. As the density, the pressure has remained constant in the boundary corners.



(a) AMR-solver.



(b) Structured solver.

Figure 40: Final solution for the density in the 2D Riemann problem.

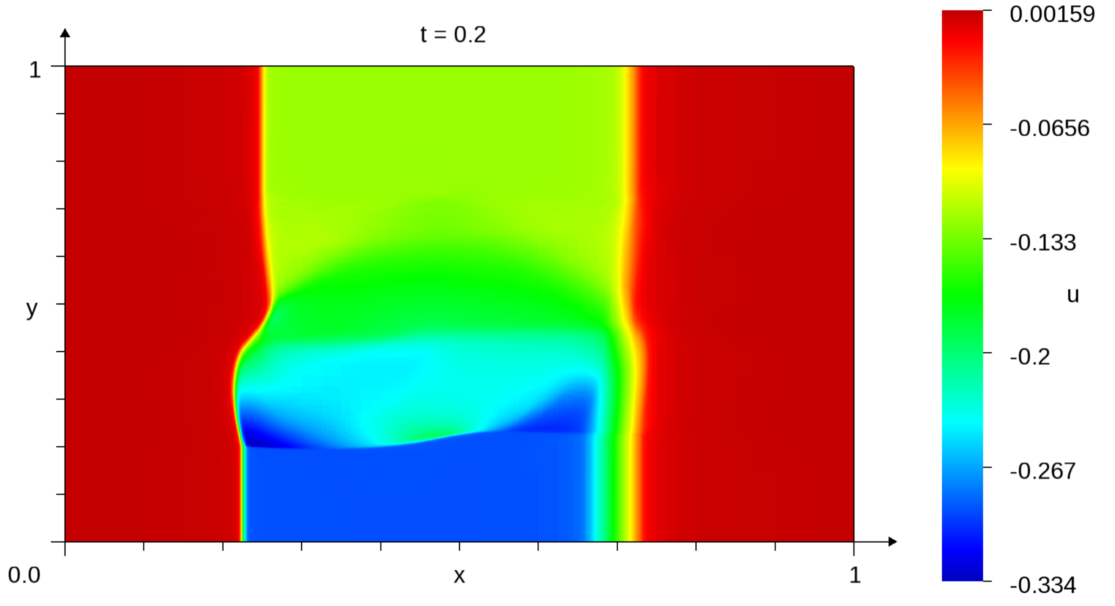


Figure 41: Final solution for the velocity component u in the 2D Riemann problem. The solution was produced by the AMR-solver.

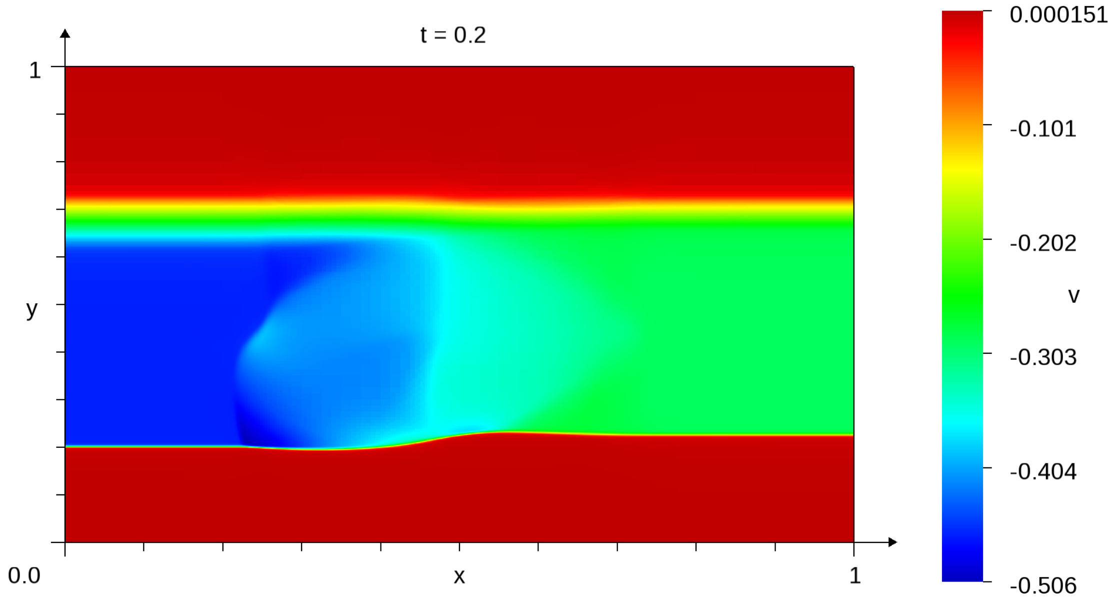


Figure 42: Final solution for the velocity component v in the 2D Riemann problem. The solution was produced by the AMR-solver.

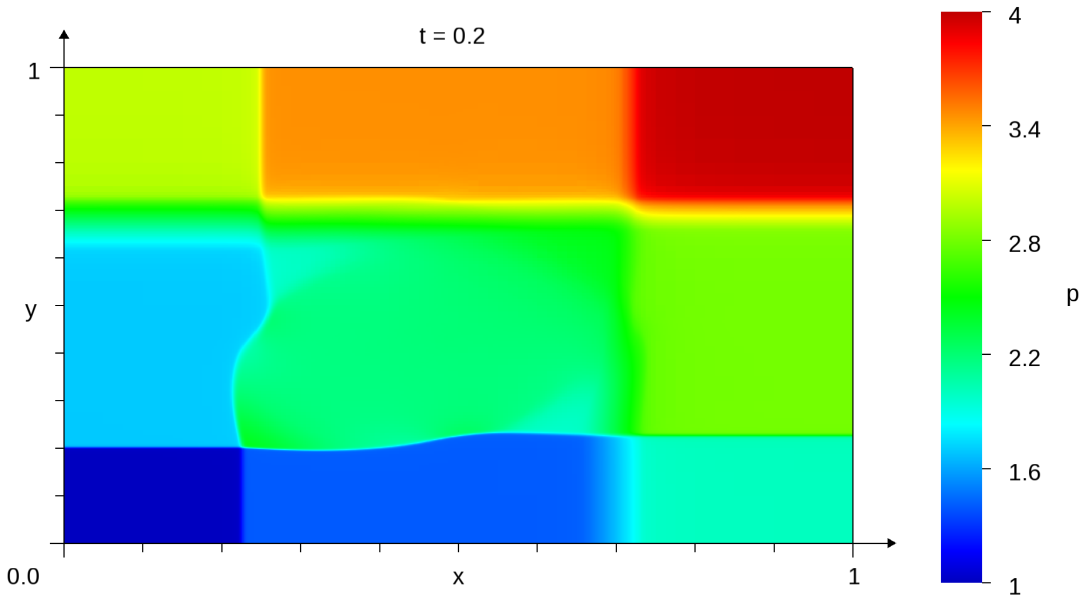


Figure 43: Final solution for the pressure in the 2D Riemann problem. The solution was produced by the AMR-solver.

Figure 44 shows the exact same density solution as Figure 40a, but with the mesh faces plotted as black lines on top. Counting refinement levels from any of the largest cells, e.g. in the corners, we can see the refinement levels $0 \leq L \leq 4$. The 2 smallest cell sizes, corresponding to refinement levels 5 and 6, are so small that the highly refined areas appear as opaque black. This occurrs at both the shock waves and both the rarefaction waves. The mesh around the contact discontinuities has also clearly been refined, but mostly to level 4. The lower part of the vertical contact discontinuity is only at level $L = 3$. For perspective, we could fit

$$\frac{(2^6)^2}{(2^3)^2} = 64 \tag{155}$$

of the smallest cells inside each of the level 3 cells. This shows a disadvantage of using absolute differences for the refine- and merge criteria: The lower part of the vertical contact discontinuity has a relatively small difference in density. It is therefore not given the highest refinement level, even though it is a discontinuity. This indicates that the refinement criterion should be based on e.g. gradients instead, to detect discontinuities with small differences.

Another observation we make from this test case is that the density ρ is a very good indicator for where high resolution is needed. For this particular test case all the high gradients and discontinuities are present for the density. We could therefore have omitted checking u and v in this case, and used ρ as the only indicator. However, as we have seen in the case with the shock reflection, cf. Figure 23 in subsection 6.1, checking ρ does not always suffice.

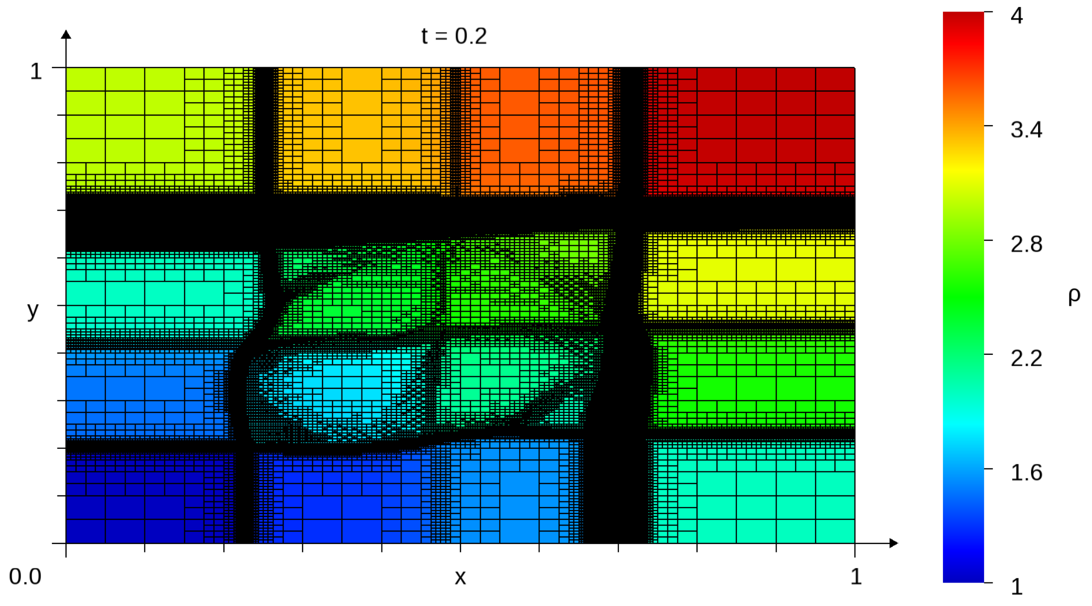


Figure 44: The final mesh used by the AMR-solver, plotted on top of the density solution.

For this test case, we will examine some technical output from the AMR-solver, and compare some of it with data from the structured solver. Table 11 shows selected values from the reports that the two solvers produced when simulating the 2D Riemann problem.

The runtime τ is the measured time that the CPU used to execute the simulation. We see that the structured solver used about twice as much time as the AMR-solver. This ratio is lower than than the values we found for the shock reflection case, indicating that the efficiency gain is a little lower for this 2D Riemann problem. The time-level-averaged number of cells $\bar{N}_i$ and the final number of cells $N_i^{n_{max}}$ are approximately 8 times higher for the structured solver. This ratio was typically much higher for the shock reflection case, as seen in Figure 34. The many waves that appear in the 2D Riemann problem require a larger portion of the domain to be well resolved. This decreases the gain from the AMR approach, as expected. Similar to the shock reflection case, we see much higher ratios in cell count than in runtime, which again indicates a potential to increase the efficiency gain considerably.

The number of time steps n_{max} tells us how many times the explicit Euler method was applied to march the solution forward. The number of mesh adaptations N_{adapt} is how many times the complete procedure of flagging, grading, refining and merging was executed. The flux/residuals time is how much of the total runtime that was spent computing time steps. It is mostly spent computing and transfering fluxes, updating primitive variables and spectral radii, computing the 1-norm and some minor book-keeping, e.g. producing the data we are currently discussing. The mesh adaptation time is the portion of the total runtime that was spent by the flagging, grading, refining and merging procedures. The first thing that these four numbers tell us is that the main time consumer is the numerical method. For the shock reflection test case this was

Table 11: Miscellaneous technical output from the AMR solver. Values from the structured solver are given where applicable, for comparison. The right column is the ratio between the left and middle columns. The meaning of each row is explained when commented in the text.

	AMR	Structured	$\frac{Structured}{AMR}$
Total CPU time τ [s]:	92.6	187	2.02
Average no. of cells $\bar{N}_i$:	200 321	1 638 400	8.18
Final no. of cells $N_i^{n_{max}}$:	206 677	1 638 400	7.93
No. of time steps n_{max}:	891	891	1
No. of mesh adaptations N_{adapt}:	133		
Flux/residuals time [s]:	75.0	(which is 81% of τ_{amr})	
Mesh adaptation time [s]:	17.6	(which is 19% of τ_{amr})	
Flagging time:	37.9 %	(of adaptation time)	
Grading time:	34.3 %	(of adaptation time)	
Refining time:	18.0 %	(of adaptation time)	
Merging time:	9.78 %	(of adaptation time)	

much more distinct, where only 3-5% of the runtime was used for mesh adaptation. Giuliani and Krivodonova [10] have conducted a similar profiling of their AMR code, where 4-45% of the CPU time was used for mesh adaptation. The 4 latter numbers also show that the average runtime spent per adaptation 132 ms, is comparable to the average time used per time step, 84 ms. Furthermore, the mesh was adapted much more often in the 2D Riemann problem,

$$\frac{N_{adapt,Riemann}}{n_{max,Riemann}} = \frac{133}{891} = 0.149 \quad , \tag{156}$$

than in the shock reflection case, cf. Table 5 and Figure 33,

$$\frac{N_{adapt,reflect}}{n_{max,reflect}} = \frac{42}{2248} = 0.0187 \quad , \tag{157}$$

where N_{adapt} is the number of adaptations and n_{max} is the number of time steps computed.

The final information we extract from Table 11 comes from the 4 bottom rows. These 4 percentages show how the mesh adaptation time is distributed among the flagging, grading, refining and merging procedures. They show that refining and merging, i.e. adding and removing cells and faces, takes much less time than flagging and grading. This is related to the linked list data structure, which allows elements to be added and removed efficiently. The refining and merging loops use only $(0.18 + 0.0978)\,0.19 = 0.053 \Rightarrow$ 5.3% of the total CPU time τ, in this simulation. Merging is more efficient than refining, because the merging loop does not need to loop over all the cells in the mesh (see subsection 5.6).

Since the numerical method is the main time consumer, future optimizations should focus on the numerical method. I.e., we should optimize the data structure to make the numerical method more efficient, even if this sacrifices some efficiency in the refining and merging loops. One optimization I propose is to store the faces in dynamically allocated arrays and keep the cells in linked lists. The face objects (see subsection 5.1) are very small in terms of bytes, and they are looped over a lot. Therefore, they can benefit from the spatial locality of an array. Conversely, the cell objects are much larger, and less frequently looped over. The size difference in bytes means that inserting and deleting faces in an array is less costly than with cells. This is further discussed in section 8.

7 Conclusions

An AMR finite volume solver using unstructured Cartesian grids has been developed for the 2D Euler equations of gas dynamics. The adaptation trigger, i.e., the criterion for *when* to adapt, is based on accumulating an approximated mass redistribution rate, cf. subsection 5.2. To the best of our knowledge, this is a new approach. The results indicate that this approach works well for both steady-state problems and transient simulations, though many more flow cases must be tested before we can assert this. For the transient test case of a 2D Riemann problem, and the development of a regular shock reflection, the new adaptation trigger keeps the mesh updated, avoiding that large gradients leave the highly resolved areas. If the resolution demanding features slow down or stop moving, the trigger activates more seldom. During convergence of the steady-state simulations this is especially evident. This adaptation trigger is also very easy to implement and inexpensive in terms of CPU time.

The benefit from the AMR approach was measured by comparing the number of cells and the CPU time of simulations that give similar errors. The cases that benefit the most from AMR are the simulations where we allow the smallest cells, i.e., the simulations with the smallest errors and highest CPU times. In these cases, the number of cells for the AMR solver was only about 5% of the number of cells for the standard solver. The efficiency gain for CPU time is much lower, but still significant. The AMR solver used 36-40% of the runtime of the standard solver to give similar errors in the most beneficial cases. The difference between these percentages is most likely related to the mesh data structure. The mesh adaptation procedures only comprise 3-20% of the AMR solver's total runtime, whereas the numerical method consumes the rest of the runtime.

The criterion for *where* to refine and coarsen the mesh was varied. This proved to have a big effect when examining runtime, number of cells and error. A relaxed criterion gives lower cell counts and runtimes and higher errors than a strict criterion, as expected. The results indicate that to maximize the benefits from this AMR approach, the refine and merge tolerances should be set depending on the smallest allowed cell size. Specifically, with smaller cells the tolerances should be smaller, and with larger cells the tolerances should be larger, to give a good balance between CPU time and error.

8 Future Outlook

More simulations should be run, with much higher cell counts. This would show if the benefit from the AMR approach continues to increase as the resolution and number of unknowns increase. If it does, then the benefit from the approach would be tremendous if it is used in supercomputing. More simulations with higher resolutions could also substatiate, or falsify, the relation between the optimal refine and merge tolerances and the smallest cell size.

The approach should be expanded to handle 3D problems, higher order approximations in time and space, and parallell computing. These expansions are necessary for the method to realize its potential. It would require several changes in the data structure and algorithms.

The adaptation trigger worked very well for the test cases in this project. We can use the approach for many different test cases to check its potential.

Evidently there is overhead related to the data structure. It would be interesting to conduct a more detailed profiling of the AMR solver to find out why its data structure slows down the numerical method so much. We could also do a detailed comparison between linked lists, trees and arrays, analyzing the complexities of the most used operations. If it has not been tested already, I propose the following combination: storing the faces in dynamic arrays, since they are small (in bytes) and looped over a lot; and storing the cell objects in a linked list, since they are large and more seldomly looped over. The faces' small sizes would increase the gain from spatial locality, and reduce the cost for insertion and deletion, in an array. Furthermore, for my previous AMR solver [1] I proposed an algorithm for handpicking and deleting multiple elements in an array and correcting invalidated pointers to the remaining elements in said array. The complexity was still higher than deleting nodes in a list, but it was much faster that deleting array elements one by one. If this algorithm can be inversed, to allow cheaper insertion (at handpicked positions in an array) as well, then we could store the faces in arrays without sacrificing the ordering scheme (see Figure 15 and subsection 5.5). The bottom line is that we could benefit from spatial locality of faces, and temporal locality of cells, when looping over faces.

The approach for ordering the interior faces, discussed in subsection 5.5, should be improved to better sustain a cache-friendly order. For the linked list approach we could use move-semantics [18] to splice broken chains and further reduce the number of cache misses.

The impaired convergence rate towards steady state should be investigated. The solution disturbances caused by the actual adaptations surly affect the convergence indicator, but we suspect there is another, more important reason for the slower convergence rate.

References

[1] F. Kristoffersen. “Adaptive Mesh Refinement for Conservation Laws”. Pre-Master project thesis. Trondheim: Department of Energy and Process Engineering, NTNU, 2018.

[2] M. J. Berger and R. J. Leveque. “Adaptive Mesh Refinement Using Wave-Propagation Algorithms for Hyperbolic Systems”. *SIAM Journal on Numerical Analysis* 35.6 (1998), pp. 2298–2316. ISSN: 0036-1429.

[3] R. Löhner. *Applied computational fluid dynamics techniques : an introduction based on finite element methods*. 2nd ed. Chichester: Wiley, 2008. ISBN: 9780470519073.

[4] G. W. Zumbusch. *Parallel multilevel methods : adaptive mesh refinement and loadbalancing*. Ed. by H. G. Bock et al. 1st ed. Wiley-Teubner series, advances in numerical mathematics. Stuttgart, Germany: Teubner, 2003. ISBN: 3-322-80063-6.

[5] T. Linde T. Plewa and V. G. Weirs. “Adaptive mesh refinement, theory and applications : proceedings of the Chicago Workshop on Adaptive Mesh Refinement Methods, Sept. 3-5, 2003”. Vol. 41. Lecture notes in computational science and engineering. Chicago Workshop on Adaptive Mesh Refinement Methods. Berlin: Springer, 2005. ISBN: 1-280-41225-9.

[6] J. A. Trangenstein. *Numerical Solution of Hyperbolic Partial Differential Equations*. Cambridge: Cambridge University Press, 2009. ISBN: 978-0-521-87727-5.

[7] O. P. Jacquotte. “Grid Optimization Methods for Quality Improvement and Adaption”. N. P. Weatherill J. F. Thompson B. K. Soni. *Handbook of Grid Generation*. Ed. by Nigel P. Weatherill Joe F. Thompson Bharat K. Soni. Boca Raton Fla: CRC Press, 1998.

[8] M. J. Aftosmis, M. J. Berger, and J. E. Melton. “Adaptive Cartesian Mesh Generation”. J. F. Thompson, B. K. Soni, and N. P. Weatherill. *Handbook of Grid Generation*. Ed. by Nigel P. Weatherill Joe F. Thompson Bharat K. Soni. Boca Raton Fla: CRC Press, 1998.

[9] Michael J. Aftosmis. *What is Cart3D?* NASA. 2018. URL: `https://www.nas.nasa.gov/publications/software/docs/cart3d/pages/cart3Dhome.html`.

[10] A. Giuliani and L. Krivodonova. “Adaptive mesh refinement on graphics processing units for applications in gas dynamics”. *Journal of Computational Physics* 381 (2019), pp. 67–90. ISSN: 0021-9991. DOI: `10.1016/j.jcp.2018.12.019`.

[11] S. Müller. *Adaptive Multiscale Schemes for Conservation Laws*. Ed. by T. J. Barth et al. Lecture Notes in Computational Science and Engineering. Berlin, Heidelberg: Springer, 2003. ISBN: 978-3-540-44325-4.

[12] M. Paszynski, D. Pardo, and V. M. Calo. “A direct solver with reutilization of LU factorizations for h -adaptive finite element grids with point singularities”. *Computers and Mathematics with Applications* 65.8 (2013), pp. 1140–1151. ISSN: 0898-1221.

[13] Randall J LeVeque. *Finite volume methods for hyperbolic problems*. eng. Cambridge, 2002.

[14] T. H. Cormen et al. *Introduction to algorithms*. 3rd ed. Cambridge, Mass.: MIT Press, 2009. ISBN: 1-62870-913-8.

[15] B. Müller. *Introduction to Computational Fluid Dynamics - Lecture notes for the course Computational Heat and Fluid Flow*. NTNU. Trondheim, 2017.

[16] J. Zierep. *Theoretische Gasdynamik*. Karlsruhe: G. Braun, 1976. ISBN: 3765020230.

[17] C. W. Schulz-Rinne, J. P. Collins, and H. M. Glaz. "Numerical Solution of the Riemann Problem for Two-Dimensional Gas Dynamics". *SIAM Journal on Scientific Computing* 14.6 (1993), pp. 1394–1414. ISSN: 1064-8275. DOI: 10.1137/0914082.

[18] B. Stroustrup. *Programming : principles and practice using C++*. 2nd ed. Upper Saddle River, N.J: Addison-Wesley Professional, 2014. ISBN: 978-0-321-99278-9.

[19] A. S. Tanenbaum and T. Austin. *Structured Computer Organization*. 6th ed. Boston: Pearson, 2013. ISBN: 978-0-273-76924-8.

Appendix A Article Submitted to Conference Proceedings, MekIT 19

The work presented in this report was also presented at the 10. National Conference on Computational Mechanics, MekIT'19. On that occation, me and my academic supervisor Bernhard Müller made an article in collaboration, for inclusion in the conference proceedings. The content in the article is based on this report and it is therefore included here, as an appendix. Page numbers, reference numbers, caption numbers and equation numbers are taken from the article, and do not follow the numbering in the rest of the report.

10. National Conference on Computational Mechanics
MekIT'19
B. Skallerud and H I Andersson (Eds)

ADAPTIVE MESH REFINEMENT ON UNSTRUCTURED CARTESIAN GRIDS FOR THE EULER EQUATIONS

FREDERIK KRISTOFFERSEN[1] AND BERNHARD MÜLLER[2]

[1] Department of Energy and Process Engineering
Norwegian University of Science and Technology
7491 Trondheim, Norway
e-mail: frederik.regi@gmail.com

[2] Department of Energy and Process Engineering
Norwegian University of Science and Technology
7491 Trondheim, Norway
e-mail: bernhard.muller@ntnu.no - Web page: http://folk.ntnu.no/bmuller/

Key words: Computational Methods, Adaptive Mesh Refinement, Euler Equations, Gas Dynamics, Shock Reflection, 2D Riemann Problem

Abstract. A finite volume solver that utilizes adaptive mesh refinement (AMR) on unstructured Cartesian grids has been developed for the two-dimensional Euler equations of gas dynamics. The solver uses the explicit Euler method and the Rusanov method for time and flux discretization, respectively. The rectangular cells can be refined by quadrisection through their centers. This preserves their aspect ratios and doubles the spatial resolution locally. Four cells that were created from the same cell refinement can be merged by reversing the refinement process. The criterion for refinement and merging is based on the absolute differences of the density and velocity components in neighboring cells. For triggering adaptation, i.e., for deciding when to perform AMR, a new criterion is proposed. It is based on accumulating the absolute rate of change of mass relative to its initial value.

The development of a regular oblique shock reflection from a plane wall is simulated, starting from an initial condition corresponding to a Riemann problem. For comparison, the Euler equations of gas dynamics are also solved by a standard finite volume solver on a structured Cartesian grid. Using AMR, the number of cells can be reduced by up to 95%, i.e. a factor 20, to achieve the same error as the standard finite volume solver. Even though mesh adaptation impairs convergence to steady state and there is some overhead related to the data structure, the AMR solver takes only 36% of the computing time needed by the standard solver, in the most beneficial cases. The potential of the AMR solver for unsteady flow is demonstrated for the simulation of a 2D Riemann problem.

1 INTRODUCTION

In computational fluid dynamics (CFD), the demand for high mesh resolution is not equal throughout the computational domain. High gradients or discontinuities in flow variables require higher resolution, i.e. smaller and more densely packed cells. It is possible to do this adaptation before simulating, if a priori knowledge, or justified assumptions are available for the flow. In many flows the regions that require high resolution are moving. Examples are vortex shedding, and moving shocks. It would be possible to resolve the vortices or shock by refining the mesh in the entire region the features traverse before they dissipate. Another approach, that can lower the computational cost, is to detect resolution demanding features while simulating, and refine or coarsen accordingly [1] [2] [3] [4]. This approach is known as *Adaptive Mesh Refinement*, or AMR for short.

An adaptation strategy describes how the AMR is executed on an algorithmic level. It should answer the questions: When, where and how to adapt?[5] The choice of adaptation strategy is closely linked with the choice of mesh, and affects how the mesh should be generated. The AMR solver developed for this project, uses an unstructured Cartesian mesh. The reason for limiting the mesh to be Cartesian, was to reduce the computation overhead related to the data structure, as explained in [6]. Marsha J. Berger has contributed to the field of AMR using unstructured Cartesian meshes in cooperation with Michael J. Aftosmis and John E. Melton [6]. This work has laid the basis for locally refining AMR softwares such as NASA's Cart3D project [7].

One method for refining a Cartesian mesh locally, is to quadrisect the cells through their centers [3]. This allows the refinement state of the cells to be classified by *levels* [8] [9]. These refinement levels L correspond to cell sizes, as shown in Figure 1. The largest permitted cell size corresponds to level $L = 0$, and gives the lowest possible resolution. The smallest allowed cell size corresponds to $L = L_{max}$, which gives the highest possible resolution. Each time L increases by one, the spatial resolution is locally doubled in all dimensions. This means that when a 2D cell is refined, it is split into four geometrically similar cells [3], as can be seen in Figure 1. In other words, all the cells have equal aspect ratio, regardless of the refinement level. For three-dimensional meshes, refined cells are split into eight cells. Only 2D cases are considered in this project.

An advantage of using this refinement approach is that one can refine very locally. Any cell can be split into four, as can be seen in Figure 1. Conversely, four cells that were created by the same cell refinement can be merged. However, it is advantageous to limit this freedom with a constraint called *grading* [8]. This constraint sets the lower limit, denoted by q, for how many cells must come between two level changes. With a grading degree of $q = 0$, there is no limit to how close two level changes can be. Essentially, this means that the refinement level can change multiple times from one cell to another, i.e. a very small cell can be adjacent to a big cell. This can negatively affect accuracy, and it can make the local refining and coarsening algorithms more intricate. With a grading degree of $q = 1$, like in Figure 1, there needs to be at least one cell between level changes.

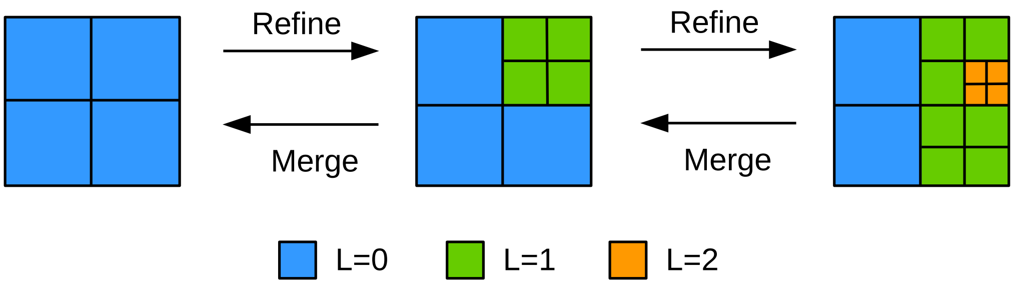


Figure 1: Principle sketch of local grid refinement and merging using quadrisection. L is the cell refinement level. The grading degree is $q = 1$, which means that between a cell at $L = 0$ and a cell at $L = 2$, there must be at least one cell at $L = 1$. This figure is taken from [10] and adapted slightly.

This means that the resolution level can change for every traversed cell, but only *one* level at a time.

Another approach of mesh refinement is known as structured AMR, or S-AMR for short [1] [6] [3]. This method involves organizing the computational domain into patches. All the cells that belong to the same patch have the same resolution. This makes the detection and refinement procedures much simpler [1], which is a an advantage. It also makes it much easier to implement methods with a higher order of accuracy in space, due to the fact that each patch is a structured sub-mesh. However, many cells will get a higher resolution than they require, unless the patches are very small. If the patches are too small, then the maximum cell size is limited, and the inter-patch communication becomes costly. The work of Marsha J. Berger must be mentioned here as well. She has contributed to the field of S-AMR, collaborating with Randall J. LeVeque. Their work in this field is utilized in the AMRCLAW software package [1], which is part the CLAWPACK package [11].

A third alternative is to refine locally using a tree structure, instead of an unstructured mesh. For a 2D solver it would be a quadtree structure. This approach allows for a relatively simple data structure if combined with the quadrisection method discussed above. However, tree data structures are more costly to traverse. Neither S-AMR or quadtrees are used in this project, but they are mentioned as alternatives.

There are multiple ways to detect where the mesh needs refinement, and where it can be coarsened. One way is to compare the entire solution from the latest time step, with a solution calculated on a coarsened mesh [1]. If the error of a cell, or patch of cells, is above a given threshold then the cell or patch is marked for refining. Another way is to evaluate the gradients in the solution, in pairs of neighbor cells. Then the cell or patch could be flagged for refinement or merging based on gradient thresholds. An even simpler approach is used in this project. We check the differences between flow variables in neighboring cells. Regardless of how the detection is done, it will consume CPU time. Therefore it is common not to detect at every time step, but to have a criterion for *when* to adapt. A simple criterion is to specify a number of time steps between each detection routine [1], or a time interval. The AMR-solver in this project decides when to adapt, by accumulating an approximated mass redistribution rate. We will elaborate this in subsection 4.1.

This paper is based on the first author's master's thesis [12], and continues with the following sections: In section 2, the 2D Euler equations and their initial and boundary conditions for two test cases involving shocks are stated. The explicit finite volume method (FVM) for the discretization of the 2D Euler equations is given in section 3. The present AMR approach based on unstructured Cartesian grids is outlined in section 4. Results for a regular oblique shock reflection from a plane wall and for a 2D Riemann problem are discussed in section 5. The accuracies and efficiencies of the AMR solver and a standard solver are compared. In section 6, conclusions are drawn.

2 GOVERNING EQUATIONS

2.1 2D Euler Equations

The 2D Euler equations of gas dynamics read:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial x} + \frac{\partial \mathbf{G}(\mathbf{U})}{\partial y} = 0 \ , \tag{1}$$

where

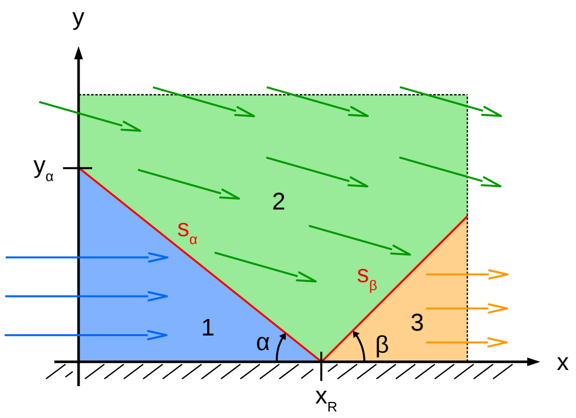
$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix} \tag{2}$$

is the vector of conservative variables,

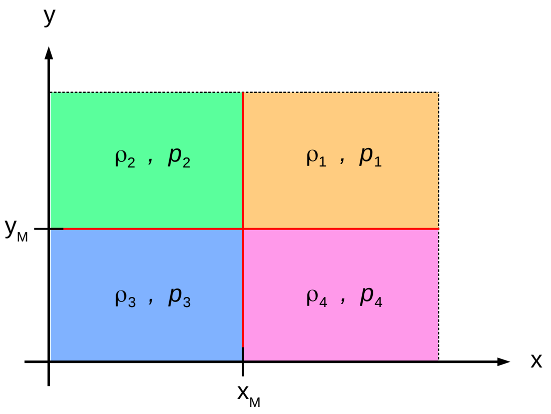
$$\mathbf{F}(\mathbf{U}) = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ u(p+\rho E) \end{bmatrix} \quad \text{and} \quad \mathbf{G}(\mathbf{U}) = \begin{bmatrix} \rho v \\ \rho v u \\ \rho v^2 + p \\ v(p+\rho E) \end{bmatrix} \tag{3}$$

are the flux vectors in the x- and y-directions, respectively. The density is denoted by ρ, the x- and y-components of the velocity by u and v, respectively, the specific total energy by E, and the pressure by p. For perfect gas considered here the pressure is related to the conserved variables by

$$p = (\gamma - 1)\left[\rho E - \frac{1}{2}\rho(u^2 + v^2)\right] \quad , \tag{4}$$



(a) Steady-state solution for the shock reflection case. The numbers denote regions 1, 2 and 3 respectively. The red lines represent the shocks. The colored arrows represent the velocity vectors.



(b) Initial conditions for the 2D Riemann problem. The red lines represent diaphragms, which are instantaneously removed at $t = 0$.

Figure 2: Principle sketches of the test cases.

using $u = \frac{\rho u}{\rho}$ and $v = \frac{\rho v}{\rho}$. γ is the constant ratio of specific heats. We shall consider air with $\gamma = 1.4$. The vector of the primitive variables is defined by:

$$\mathbf{V} = \begin{bmatrix} \rho \\ u \\ v \\ p \end{bmatrix} \tag{5}$$

2.2 Initial and Boundary Conditions

We will examine two test cases: regular shock reflection, cf. Figure 2a, and a 2D Riemann problem, cf. Figure 2b.

The shock reflection case seen in Figure 2a, is a steady-state problem. The figure shows how the steady solution can be divided into regions 1, 2 and 3. Inside each region, the flow variables are constant:

$$\mathbf{V}(x,y) = \begin{cases} \mathbf{V}_1 & \text{for} \quad x < x_R \quad \text{and} \quad y < s_\alpha(x) \\ \mathbf{V}_2 & \text{for} \quad y > s_\alpha(x) \quad \text{or} \quad y > s_\beta(x) \\ \mathbf{V}_3 & \text{for} \quad x > x_R \quad \text{and} \quad y < s_\beta(x) \end{cases} \tag{6}$$

where subscripts 1, 2 and 3 denote regions illustrated in Figure 2a. $s_\alpha(x)$ and $s_\beta(x)$ are the shock graphs in the same figure. $\mathbf{V}_1$ describes the uniform incoming flow at $M_1 > 1$.

$\mathbf{V}_2$ describes the flow behind the oblique shock s_α with shock angle α. $\mathbf{V}_3$ describes the flow behind the reflected shock s_β. The analytical solution for the variables in (6) is given in [12].

We use the following initial conditions (IC) for the shock reflection case:

$$\mathbf{V}(x, y, t = 0) = \begin{cases} \mathbf{V}_1 & \text{for} \quad y < y_\alpha \\ \mathbf{V}_2 & \text{for} \quad y > y_\alpha \end{cases} \tag{7}$$

where y_α is shown in Figure 2a. We use the following boundary conditions (BC) for this case:

$$\mathbf{V}(x = 0, y, t) = \begin{cases} \mathbf{V}_1 & \text{for} \quad y < y_\alpha \\ \mathbf{V}_2 & \text{for} \quad y > y_\alpha \end{cases} \tag{8}$$

$$\mathbf{V}(x, y = y_{max}, t) = \mathbf{V}_2 \tag{9}$$

$$v(x, y = 0, t) = 0 \tag{10}$$

$$\left.\frac{\partial \mathbf{U}}{\partial x}\right|_{x = x_{max}} = 0 \tag{11}$$

The second test case is a 2D Riemann problem. This case is defined by its initial condition:

$$\mathbf{V}(x, y, t = 0) = \begin{cases} \mathbf{V}_1 & \text{for} \quad x > x_M \; , \quad y > y_M \\ \mathbf{V}_2 & \text{for} \quad x < x_M \; , \quad y > y_M \\ \mathbf{V}_3 & \text{for} \quad x < x_M \; , \quad y < y_M \\ \mathbf{V}_4 & \text{for} \quad x > x_M \; , \quad y < y_M \end{cases} \tag{12}$$

which is illustrated in Figure 2b. Subscripts 1-4 in (12) denote the regions in the latter figure. Physically, we envision this IC being achieved by diaphragms. At time $t = 0$ the diaphragms are removed, causing different types of waves to interact with each other. We will see this in the results.

The artificial boundaries are treated by using homogeneous Neuman boundary conditions, i.e.

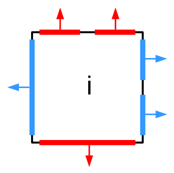
$$\frac{\partial \mathbf{U}}{\partial x} = 0 \quad \text{at} \quad x = 0 \quad \text{and} \quad x = x_{max} \tag{13}$$

$$\frac{\partial \mathbf{U}}{\partial y} = 0 \quad \text{at} \quad y = 0 \quad \text{and} \quad y = y_{max} \tag{14}$$

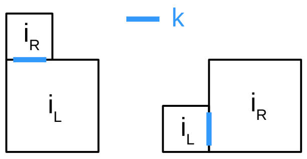
for the domain $[0\ ,\ x_{max}] \times [0\ ,\ y_{max}]$.

3 NUMERICAL APPROACH

In this section the numerical method is outlined.



(a) The cell i is in focus. It has 6 enclosing faces, whose normal vectors point away from the cell i. The vertical faces $k_{F,i}$ are colored blue, and the horizontal faces $k_{G,i}$ are colored red. There can be one or two enclosing faces on each side, giving a minimum of 4 and a maximum of 8 faces in total.



(b) Two examples where a face k is in focus. Its adjacent cells are denoted by subscripts L and R. For vertical faces the subscripts denote cells on the left and right side of the face, respectively. For horizontal faces the subscripts L and R denote cells below and above the face, respectively.

Figure 3: Notation for cells and faces, depending on focus.

3.1 Spatial Discretization

We use Cartesian grids with rectangular cells. This means that all cell faces are either vertical or horizontal. We also set the cell sizes $\Delta x = \Delta y$ for all cells, meaning that all cells are squares. The finite volume method (FVM) is used to approximate the cell averages of the conserved variables.

The fluxes (3) are approximated at the cell faces by the Rusanov method, also known as the local Lax-Friedrichs method. The flux approximations at a face k are:

$$\mathbf{F}_k = \frac{1}{2}\left[(\mathbf{F}(\mathbf{U}_R) + \mathbf{F}(\mathbf{U}_L)) - a_k(\mathbf{U}_R - \mathbf{U}_L)\right] \quad , \tag{15}$$

$$\mathbf{G}_k = \frac{1}{2}\left[(\mathbf{G}(\mathbf{U}_R) + \mathbf{G}(\mathbf{U}_L)) - b_k(\mathbf{U}_R - \mathbf{U}_L)\right] \quad , \tag{16}$$

where $\mathbf{F}_k$ and $\mathbf{G}_k$ are the flux approximations at the vertical and horizontal faces, respectively. Subscripts L, R and k are explained in Figure 3b. a_k and b_k are determined by the spectral radii of the Jacobian matrices $\frac{\partial \mathbf{F}(\mathbf{U})}{\partial \mathbf{U}}$ and $\frac{\partial \mathbf{G}(\mathbf{U})}{\partial \mathbf{U}}$, respectively, in the cells i_L and i_R.

$$a_k = \max\{|u_L| + c_L \ , \ |u_R| + c_R\} \quad , \quad b_k = \max\{|v_L| + c_L \ , \ |v_R| + c_R\} \quad , \tag{17}$$

where $c = \sqrt{\gamma p/\rho}$ is the speed of sound.

3.2 Time Discretization

The discretization in time is done by the explicit Euler method

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n - \frac{\Delta t^n}{V_i}\left(\sum_{k\in k_{F,i}}[F_k^n A_k \hat{x}_k] + \sum_{k\in k_{G,i}}[G_k^n A_k \hat{y}_k]\right) \tag{18}$$

where i is a cell index, superscript n denotes time level, V_i is the volume of the cell i, $k_{F,i}$ contains the vertical faces enclosing cell i, and $k_{G,i}$ contains the horizontal faces enclosing cell i, as shown in Figure 3a. A_k is the area of face k, $\hat{x}_k$ and $\hat{y}_k$ are the x- and y-components, respectively, of the normal vector of face k, which always points away from cell i, as shown in Figure 3a. The time step size Δt^n is set at every time level to give the Courant number

$$C_{max} = \frac{\max_i\{|u_i| + c_i\}\Delta t}{\min_i\{\Delta x_i\}} + \frac{\max_i\{|v_i| + c_i\}\Delta t}{\min_i\{\Delta y_i\}} \leq 1 \quad . \tag{19}$$

3.3 Stopping Criterion for the Steady-State Solution

Steady state is assumed if

$$\left\|\frac{\Delta\rho^n}{\Delta t^n}\right\|_1 \leq 10^{-4}\left\|\frac{\Delta\rho^0}{\Delta t^0}\right\|_1 \quad , \tag{20}$$

where

$$\|\Delta\rho^n\|_1 = \sum_i\left[V_i\left|\rho_i^{n+1} - \rho_i^n\right|\right] \tag{21}$$

is the 1-norm of the density change, corresponding to the sum of the modulus of the mass change in all cells i, from time level n to time level $n+1$.

$$\left\|\frac{\Delta\rho^n}{\Delta t^n}\right\|_1 = \frac{\|\Delta\rho^n\|_1}{\Delta t^n} \tag{22}$$

approximates the rate of mass redistribution between time levels n and $n+1$. It is required to be lower equal to its initial value by the factor 10^{-4}.

4 AMR APPROACH

In this section we will see how the mesh is adapted in 4 sub-procedures: Flagging, grading, refining and merging. The data structure of the AMR-solver will not be discussed here, but can be found in [12]. We note that the cell- and face objects are stored in doubly linked lists.

4.1 Adaptation Trigger

An adaptation strategy should specify when, where and how to adapt [5]. The answer to the *when* is what will be referred to as the adaptation trigger. When the trigger activates, the flagging, grading, refining and merging procedures are run, in that order. The trigger is related to the mass redistribution rate $\|\Delta\rho^n\|_1 / \Delta t^n$, as defined in (22) and (21).

The first mesh adaptations are triggered at time level $n_0 = 0$, i.e., based on the initial condition (IC). The number of initial adaptations is equal to the upper limit L_{max} for the refinement level. Thus, if there are large gradients or discontinuities in the IC, the nearby cells will be refined to the highest level before the first time step starts. After an adaptation, say at $n = n_a$, the subsequent values of $\|\Delta\rho^n\|_1 / \Delta t^n$ are accumulated until

$$\sum_{n=n_a}^{n_{a+1}-1} \frac{\|\Delta\rho^n\|_1}{\Delta t^n} \geq 5\, \frac{\|\Delta\rho^0\|_1}{\Delta t^0} \tag{23}$$

Then, mesh adaptation is triggered at time level $n = n_{a+1}$. The factor 5 above was found by trial and error and may be case specific. However, it has given good results for both test cases in all the simulations this far.

4.2 Flagging

The flagging procedure is the first step of each mesh adaptation. It decides *where* the mesh will be refined, and where it can be coarsened. To decide this, a refine criterion, based on absolute differences in the flow variables ρ, u and v is used. These 3 variables will be referred to as the indicators.

The flagging procedure is a loop over all the interior faces. For each face k we check the absolute difference in all the indicators, between the 2 adjacent cells.

$$\Delta\rho_k = |\rho_R - \rho_L| \quad , \quad \Delta u_k = |u_R - u_L| \quad , \quad \Delta v_k = |v_R - v_L| \tag{24}$$

where subscripts R and L denote values in the right and left adjacent cells if the face is vertical. For horizontal faces, the subscripts R and L denote the cells above and below the face, respectively, like in Figure 3b. By looping over interior faces we check all combinations of adjacent cell pairs exactly once. The differences (24) are compared to the refine thresholds

$$\delta_{r,\rho} = \eta_r\, \Delta\rho_{max} \quad , \quad \delta_{r,u} = \eta_r\, \Delta u_{max} \quad , \quad \delta_{r,v} = \eta_r\, \Delta v_{max} \tag{25}$$

and the merge thresholds

$$\delta_{m,\rho} = \eta_m\, \Delta\rho_{max} \quad , \quad \delta_{m,u} = \eta_m\, \Delta u_{max} \quad , \quad \delta_{m,v} = \eta_m\, \Delta v_{max} \tag{26}$$

where

$$\begin{aligned}\Delta\rho_{max} &= \max_{i,0\leq m\leq n}\{\rho_i^m\} - \min_{i,0\leq m\leq n}\{\rho_i^m\}\\ \Delta u_{max} &= \max_{i,0\leq m\leq n}\{u_i^m\} - \min_{i,0\leq m\leq n}\{u_i^m\}\\ \Delta v_{max} &= \max_{i,0\leq m\leq n}\{v_i^m\} - \min_{i,0\leq m\leq n}\{v_i^m\}\end{aligned} \tag{27}$$

are the largest spreads for all the indicators, evaluated over all cells i and all time levels m until n. That is, $\max_{i,0\leq m\leq n}\{\rho_i^m\}$ is the largest density that has existed in any cell i at any time level m until time level n. Conversely, $\min_{i,0\leq m\leq n}\{\rho_i^m\}$ is the lowest cell density until time level n. The same applies for the other indicators.

η_r is a multiplier that affects all the refine thresholds (25). It sets the thresholds δ_r as fractions of the largest spreads (27). We will refer to η_r as the refine tolerance. In a similar way, η_m sets the *merge* thresholds δ_m in (26), also as fractions of the largest spreads (27). η_m will be referred to as the merge tolerance. Both these tolerances are given as parameters for the AMR solver to adjust the strictness of the refinement criterion.

The comparisons between absolute differences and thresholds are used to flag the cells in the following way:

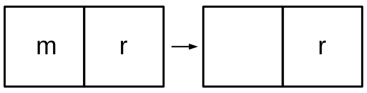
- Cells are flagged for refinement if *any* of the indicators has large absolute differences, over *any* of the enclosing faces, *and* the cell is not at the maximum refinement level L_{max}. I.e. cell i with refinement level $L_i < L_{max}$ is flagged for refinement, if at least one of its enclosing faces k has $\Delta\rho_k > \delta_{r,\rho}$ or $\Delta u_k > \delta_{r,u}$ or $\Delta v_k > \delta_{r,v}$.
- Cells are flagged for merging if *all* of the indicators have small absolute differences, over *all* of the enclosing faces, *and* the cell is not at the minimum refinement level 0. I.e. cell i with refinement level $L_i > 0$ is flagged for merging, if all of its enclosing faces k have $\Delta\rho_k < \delta_{m,\rho}$ and $\Delta u_k < \delta_{m,u}$ and $\Delta v_k < \delta_{m,v}$.
- No cell has both the refine flag and the merge flag. They have either one of them, or none of them.

4.3 Grading

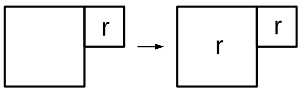
As discussed in the introduction, we will use mesh grading to prevent adjacent cells to be more than 1 refinement level apart. Thus, we require

$$\Delta x_L \in \left\{ \frac{1}{2}\Delta x_R \ , \ \Delta x_R \ , \ 2\,\Delta x_R \right\} \quad , \tag{28}$$

at all interior faces, where Δx_L and Δx_R are the cell sizes of the adjacent cells. To achieve it we adjust the flags in a another loop over interior faces, which we call the grading loop. Examples of this are shown in Figure 4. We limit the number of combinations to handle, by assuming that (28) holds when the grading loop starts.



(a) Merge flag is removed.



(b) Refine flag is added.

Figure 4: Examples of illegal flag and level combinations, and following countermeasures to avoid size ratio 4. m and r denote the merge and refine flags. The figures are taken from [10], with small changes.

If the adjacent cells are at equal levels $L_L = L_R$ there is only one illegal flag combination, which is shown in Figure 4a. The figure also shows how it is solved. If the cells are at different levels, there are multiple illegal flag combinations that call for different countermeasures. These are given in [12] and so are the implementation details.

4.4 Refining and Merging

The refining and merging of cells are done separately in two loops. The refine loop is a loop over all the cells in the mesh, where we check the refine flag r_i for each cell i. If $r_i = 1$ we split cell i into four new cells, where the flow variables are copied from the refined cell. Similarly, the merge loop checks the merge flag m_i for all cells i. However, cells are only allowed to merge if all the four cells that came from one cell refinement have merge flags. When four cells merge, the flow variables in the resulting cell is set to the average from the merging cells, i, $i+1$, $i+2$ and $i+3$,

$$\frac{\mathbf{U}_i + \mathbf{U}_{i+1} + \mathbf{U}_{i+2} + \mathbf{U}_{i+3}}{4} \quad . \tag{29}$$

This changes the solution, but still conserves $\mathbf{U}$. The implementation of the refine and merge loops are detailed in [12].

5 RESULTS AND DISCUSSION

In this section, the obtained results will be presented and discussed.

A simple structured solver was implemented to act as a basis for comparison. This solver uses the same numerical approach as the AMR-solver, but its data structure is based on arrays. The mesh in this solver will always be structured and equidistant and cannot be adapted. However, the grid spacing can be varied from one simulation to another. This was done to provide a justified comparison that will show the potential overhead of the AMR data structure.

5.1 Development of Shock Reflection

The shock reflection test case was introduced in subsection 2.2. We set the incoming Mach number $M_1 = 2.5$ and the shock angle $\alpha = \frac{\pi}{5}$. In this subsection we will examine how

Table 1: Parameters and derived parameters used by the AMR-solver to simulate the development of the shock reflection. M_1 is the Mach number in the uniform incoming flow, and α is the incoming shock angle, cf. Figure 2a. Δx_{max} is the largest allowed cell size corresponding to the lowest refinement level $L = 0$. L_{max} is the highest permitted refinement level. η_r and η_m are the refine and merge tolerances, respectively, defined in subsection 4.2.

(a) Given parameters.

M_1 :	2.5
α :	$\frac{\pi}{5}$
Δx_{max} :	$\frac{1}{10}$
L_{max} :	4
η_r :	0.04
η_m :	0.0182

(b) Available refinement levels and corresponding cell sizes. These can always be derived from Δx_{max} and L_{max}.

					L_{max}
L :	0	1	2	3	4
Δx_{amr} :	$\frac{1}{10}$	$\frac{1}{20}$	$\frac{1}{40}$	$\frac{1}{80}$	$\frac{1}{160}$
	Δx_{max}				Δx_{min}

the shock reflection flow develops from the initial condition to the steady-state solution. The main reasons for studying this are:

1. To check that the mesh adapts as intended by checking whether the fine resolution areas follow the shocks.
2. To provide intuitive understanding of the flow.
3. To control that the solution actually converges and achieves a steady state.

We will primarily study the results from a simulation by the AMR-solver, with parameters as shown in Table 1. The allowed refinement levels and cell sizes are listed in Table 1b. These are driven parameters, dictated by Δx_{max} and L_{max}. For later simulations, only Δx_{max}, L_{max} and Δx_{min} will be listed. All the results displayed in this subsection will be from the AMR simulation with the parameters in Table 1, unless something else is specified.

Figures 5 and 6 are cell plots of the density ρ and the velocity component u. They show the development of these variables, from their initial conditions in figures 5a and

5b, until steady state in figures 6c and 6d. For both ρ and u we see that the incoming shock moves towards the lower boundary wall, but with a higher speed for ρ than for u. Since the refinement criterion discussed in subsection 4.2 is based on differences (24) of ρ, u and v, we see two distinct branches of highly refined cells. The reflected shocks start to form in figures 5e and 5f. These figures, and figures 6a and 6b show that the reflected shocks form differently for ρ than for u. The converged solutions are shown in figures 6c and 6d, where the shock locations for ρ and u coincide. The developments of the second velocity component v and the pressure p are not plotted in this paper, but are examined in [12]. For v and p, the shock locations are almost identical to the shock location for the density ρ, during the entire development.

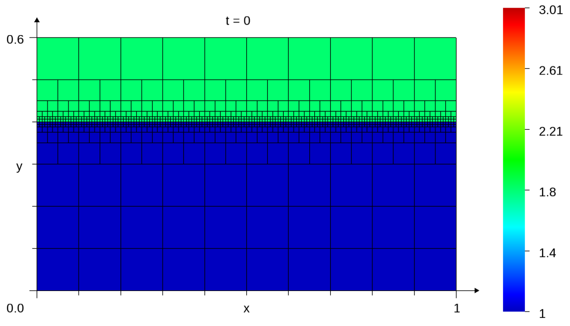
To check that the solutions discussed above achieve steady states, the convergence history of the AMR simulation cf. Table 1 is plotted in Figure 7. Another simulation was executed, by the standard finite volume solver solver, using a grid spacing $\Delta x_{str} = \Delta x_{min}$, i.e., equal to the smallest cell size allowed in the AMR-simulation. This standard simulation is also included in the convergence history plot in Figure 7. In said figure, the red graph shows that the convergence indicator $\|\Delta \rho^n\|_1 / \Delta t^n$, scaled by its initial value, decreases monotonically for the structured solver, until reaching the threshold, 10^{-4}. For the AMR simulation the convergence indicator is indicated by the blue graph in Figure 7. It jumps up at the time levels indicated by the dashed vertical lines, because the mesh is adapted at those times, causing disturbances of the numerical solution. In this particular case, the convergence threshold is reached at $n = 2248$ for the AMR solver, and at $n = 1671$ for the standard solver. This means that the AMR-solver takes 1.35 times as many time steps as the structured solver, which is a recurring trend. Although the factor varies between 1.3 and 1.6, the AMR-solver required more time steps to converge in all simulations.

Figure 7 makes yet another important point, if we observe the spacing between the dashed gray lines. It is evident that the mesh adaptations happen much more seldom later in the simulation. This relates to the adaptation trigger that was introduced in subsection 4.1. As $\|\Delta \rho^n\|_1 / \Delta t^n$ decreases, more time steps are required before mesh adaptation is triggered.

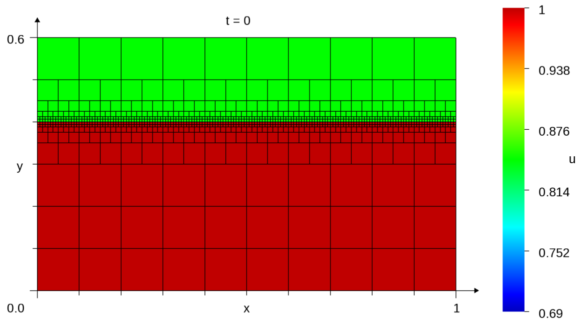
5.2 Comparing Error versus Runtime

To measure the gains from the AMR approach, we will compare the runtime, error and number of cells in multiple simulations. These simulations are grouped into 6 series, where 1 series was executed by the structured solver and the remaining 5 series were simulated by the AMR-solver. In each of the 5 AMR series, there are 11 simulations, giving a total of 55 AMR simulations. Inside the series, the parameters that control the available cell sizes are varied according to Table 2.

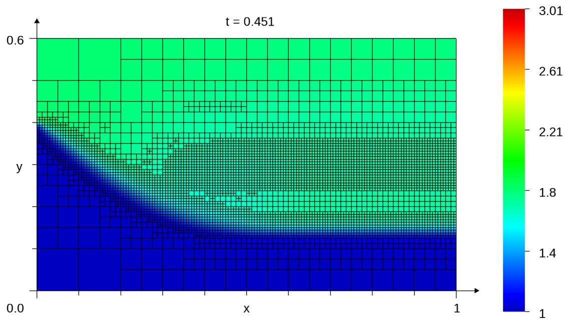
The 5 AMR series use different tolerances for the refine- and merge criteria that were introduced in subsection 4.2. The tolerances are given in Table 3, were the merge tolerances have been set as $\eta_m = \frac{\eta_r}{2.2}$. The first series has high tolerances. This gives relaxed



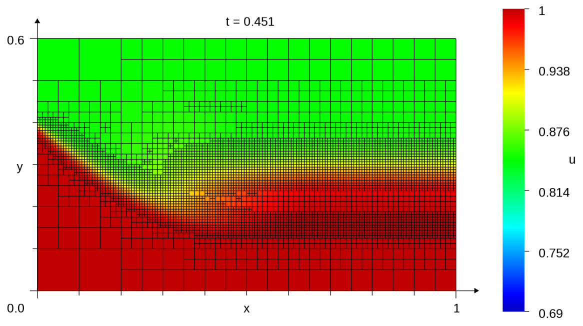
(a) IC for density.



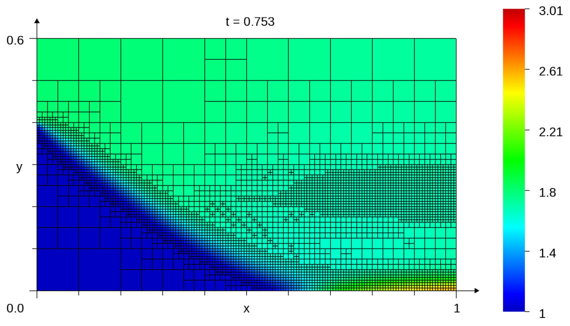
(b) IC for velocity component u.



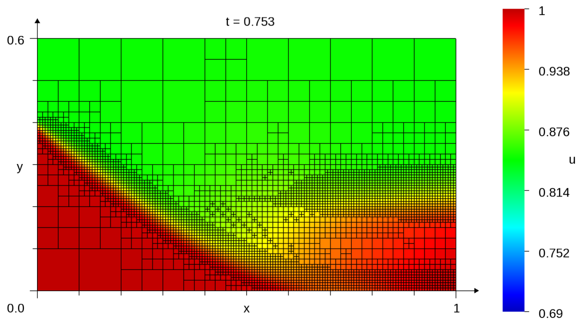
(c) Density. The shock moves downwards, and the area around it has high resolution.



(d) Velocity component u. The discontinuity moves downwards, but slower than for ρ. Thus there are two branches of highly refined cells.

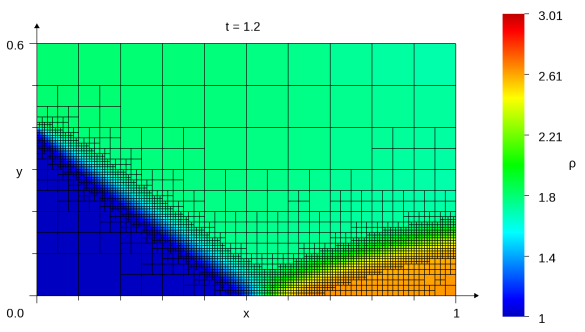


(e) Density. The shock has reached the wall, and starts to reflect.

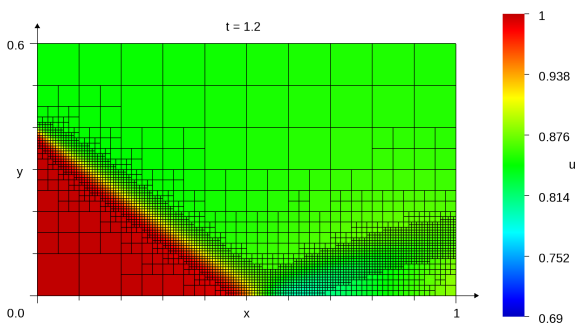


(f) Velocity component u. The left part of the wave has reached the wall. The right part has become diffuse and exits through the right boundary.

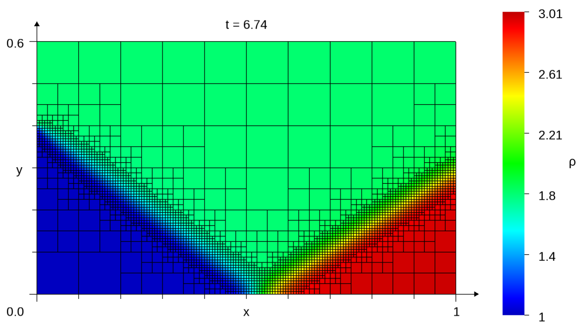
Figure 5: Cell plots of the developments of density ρ and velocity component u, from initial conditions until the shock reaches the wall at $y = 0$. The black lines are the faces in the mesh. The most highly refined cells, at level $L = L_{max}$, follow the discontinuities of both ρ and u.



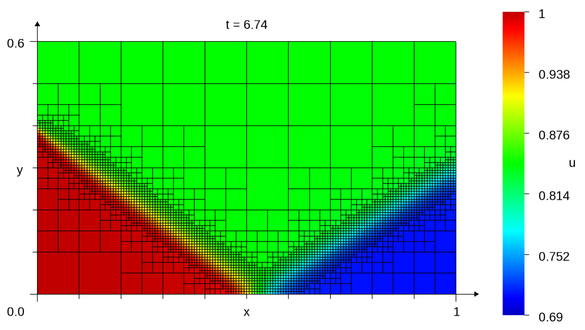
(a) Density. The reflected shock is bending upwards, while the reflection point moves leftwards.



(b) Velocity component u. The reflected shock forms extensionally, like a telescope, while bending upwards.



(c) Converged solution for the density.



(d) Converged solution for the velocity component u.

Figure 6: Cell plots of the developments of density ρ and velocity component u, from time $t = 1.2$ until steady state, at $t = 6.74$. The black lines are the faces in the mesh. The most highly refined cells, at level $L = L_{max}$, follow the discontinuities of both ρ and u.

Table 2: Parameters that vary within an AMR simulation series. Δx_{max} is the size of all the cells when the mesh is created and L_{max} is the highest permitted refinement level. Δx_{min} is the smallest permitted cell size, and follows directly from Δx_{max} and L_{max}.

Sim. number:	1	2	3	4	5	6	7	8	9	10	11
Δx_{max} :	$\frac{1}{10}$	$\frac{1}{15}$	$\frac{1}{10}$	$\frac{1}{15}$	$\frac{1}{10}$	$\frac{1}{15}$	$\frac{1}{10}$	$\frac{1}{15}$	$\frac{1}{10}$	$\frac{1}{15}$	$\frac{1}{20}$
L_{max} :	2	2	3	3	4	4	5	5	6	6	6
Δx_{min} :	$\frac{1}{40}$	$\frac{1}{60}$	$\frac{1}{80}$	$\frac{1}{120}$	$\frac{1}{160}$	$\frac{1}{240}$	$\frac{1}{320}$	$\frac{1}{480}$	$\frac{1}{640}$	$\frac{1}{960}$	$\frac{1}{1280}$

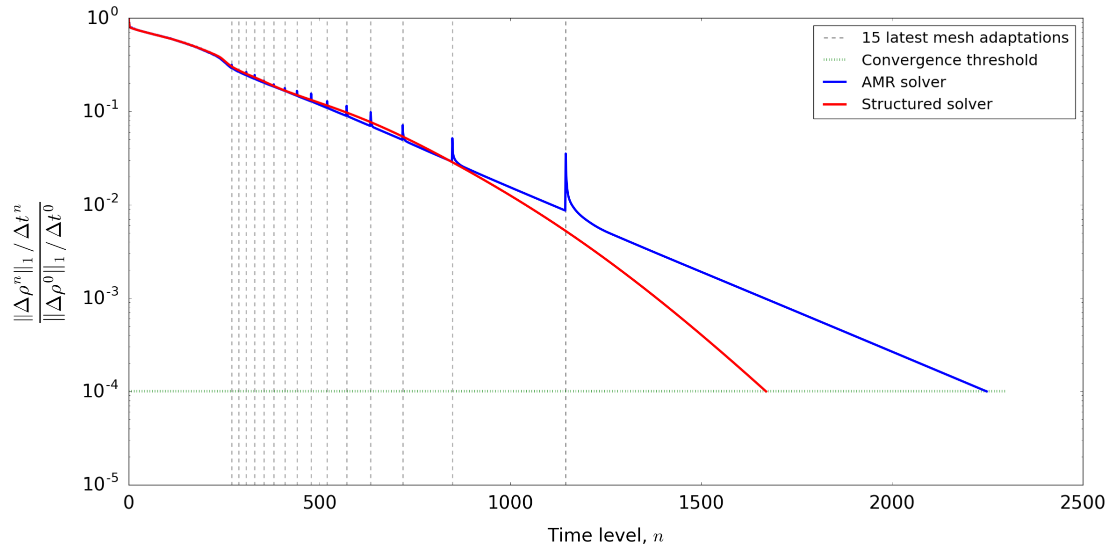


Figure 7: Convergence history plot with logarithmic ordinate axis. The red and blue graphs show the development of the convergence indicator $\|\Delta\rho^n\|_1/\Delta t^n$, normalized by its initial value. The dashed gray vertical lines show the time levels when the last 15 of the total 42 adaptations occurred. Both solvers used $\Delta x_{min} = \Delta y_{min} = \frac{1}{160}$.

refine and merge criteria, meaning that fewer cells will be refined and more cells will merge. The last series has low tolerances. This gives stricter criteria, forcing more cells to be refined and permitting fewer cells to merge.

The structured series consist of 13 simulations executed by the structured solver using equidistant grids. The 13 simulations use different mesh resolutions listed as grid spacings in Table 4. These chosen cell sizes are similar to Δx_{min} in Table 2, which was intended to give errors in the same range. The error considered here is the volume averaged density error

Table 3: Refinement and merging tolerances for the 5 AMR series. η_r and η_m are defined in subsection 4.2.

Series no.:	η_r	η_m
1	0.08	0.0364
2	0.06	0.0273
3	0.04	0.0182
4	0.02	0.0091
5	0.01	0.0045

Table 4: Cell sizes, i.e. grid spacings for the structured solver.

Sim. number:	1	2	3	4	5	6	7	8	9	10	11	12	13
Δx :	$\frac{1}{35}$	$\frac{1}{40}$	$\frac{1}{60}$	$\frac{1}{80}$	$\frac{1}{120}$	$\frac{1}{160}$	$\frac{1}{240}$	$\frac{1}{320}$	$\frac{1}{480}$	$\frac{1}{640}$	$\frac{1}{960}$	$\frac{1}{1100}$	$\frac{1}{1200}$

$$\varepsilon_\rho = \frac{1}{V_D} \left\|\rho - \bar{\rho}_{exact}\right\|_1 \tag{30}$$

where $V_D = \sum_i V_i$ is the domain volume and

$$\bar{\rho}_{exact,i} = \frac{1}{V_i} \int\limits_{\Omega_i} \rho_{exact}(x, y)\mathrm{dxdy} \tag{31}$$

is the exact average density in cell i.

From these 55 AMR simulations and 13 standard simulations, the computation times T_R, density errors ε_ρ, and cell counts N_{cell} were logged and presented in plots. In Figure 8 we see that the AMR solver uses much fewer cells than the standard solver to achieve the same error. The AMR simulation with the lowest error (leftmost blue circle) used only about 5% of the cells the standard solver used. This simulation also showed a large saving in runtime, using only about 36% of the time used by the standard solver. The comparisons of runtime versus error is shown in Figure 9. In this plot the runtimes were normalized as

$$T_R^* = \frac{T_R}{T_{R,str}} \quad , \tag{32}$$

where T_R is the runtime the simulation in focus, and $T_{R,str}$ is the runtime of a simulation from the structured solver, that gives the same error as the simulation in focus. Since no two simulations have equal errors, $T_{R,str}$ is found by interpolation between structured simulations:

$$T_{R,str} = T_R^{right} + (\varepsilon_\rho - \varepsilon_\rho^{right}) \frac{T_R^{left} - T_R^{right}}{\varepsilon_\rho^{left} - \varepsilon_\rho^{right}} \quad , \tag{33}$$

where the superscripts ^{left} and ^{right} denote the closest structured solution on each side, in terms of error. The most obvious trend in Figure 9 is that the AMR solver mostly uses less runtime to achieve the same error as the standard solver. However, the red graph shows that if the tolerances are too relaxed, we will not achieve this gain in the left part of the plot. Even though the high tolerance $\eta_r = 0.08$ gives low runtimes and cell counts, it increases the error so much, that the standard solver could achieve the same error using less runtime. Comparing the AMR series we see that the series with the highest tolerance gets the largest runtime gain in the right part of the plot. The simulations in

Table 5: Comparison between an AMR simulation and a standard simulation, where we see a large benefit from AMR. For the standard solver the refine tolerance η_r is not applicable, and Δx_{min} was the size of *all* the cells. N_{cell} is the final number of cells in the mesh. ε_ρ is the density error. The ratios are the values from the standard solver divided by the AMR values.

	η_r	Δx_{min}	N_{cell}	T_R $[s]$	ε_ρ
AMR:	0.01	$\frac{1}{1280}$	35 433	327	0.00742
Standard:		$\frac{1}{1100}$	726 000	903	0.00735
Ratio:			20.5	2.76	0.991

the right part of the plot have larger cells overall and thus higher errors. Conversely, small tolerances giving strict criteria, seem to give lower gain with fewer larger cells, and the highest gains with smaller cells, i.e. in the left part of the figure, where the errors are lower. If this relationship holds, the refine and merge tolerances should be set depending on the smallest allowed cell size.

5.3 Simulation of a 2D Riemann Problem

The second test case that was studied is a 2D Riemann problem. It was introduced in subsection 2.2. For this flow case we do not have an analytical solution to verify the solution. However, there are other gains from simulating this test case:

1. We will check that the AMR-solver finds the same solution as the structured solver.
2. We will check that the mesh adaptation algorithm detects high gradients in a more complicated flow.
3. We will still compare runtimes between the AMR-solver and the structured solver.

For the initial conditions given in equation (12) and Figure 2b we set the following numerical values:

$$\rho_1 = p_1 = 4 \quad , \quad \rho_2 = p_2 = 3 \quad , \quad \rho_3 = p_3 = 1 \quad , \quad \rho_4 = p_4 = 2 \tag{34}$$

$$u(x,y,t=0) = v(x,y,t=0) = 0 \tag{35}$$

The parameter settings that were used to simulate this test case with the AMR-solver are given in Table 6. The refine and merge tolerances η_r and η_m, and the smallest cell size $\Delta x_{min} = \Delta y_{min}$ are set relatively small, if compared to the parameters used in subsection 5.2. This was done to ensure that all the different wave types were resolved. For the structured solver, the uniform grid spacing Δx_{str} was set equal to the smallest

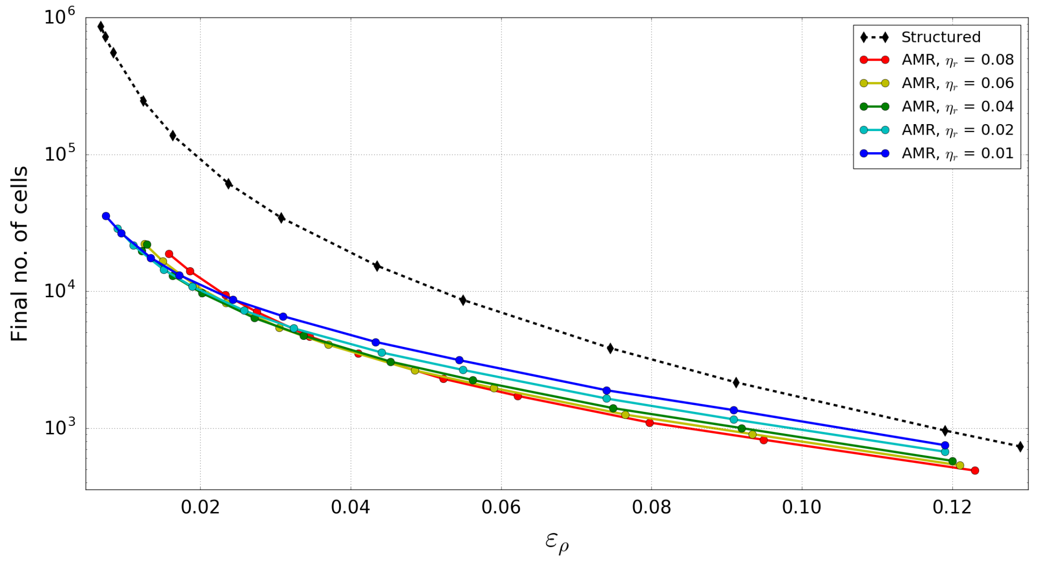


Figure 8: Number of mesh cells N_{cell} plotted versus the density error ε_ρ. Ordinate axis is logarithmic.

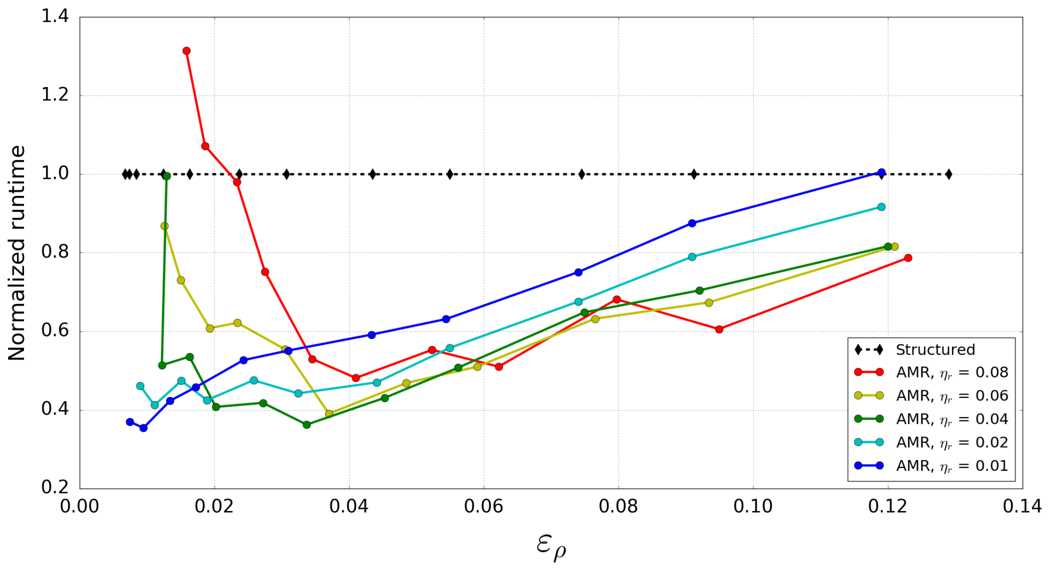


Figure 9: Normalized runtime T_R^* plotted versus density error ε_ρ. The scaling factor for the runtimes are the runtimes from the structured simulations. Therefore $T_R^* = 1$ for all structured simulations.

Table 6: Parameters used by the AMR-solver to simulate the 2D Riemann problem.

Δx_{max} :	$\frac{1}{20}$
L_{max} :	6
Δx_{min} :	$\frac{1}{1280}$
η_r :	0.015
η_m :	0.0068

allowed cell size Δx_{min} for the AMR-solver. We limit the test case to the time $0 \leq t \leq 0.2$, before any of the waves reach the domain boundaries.

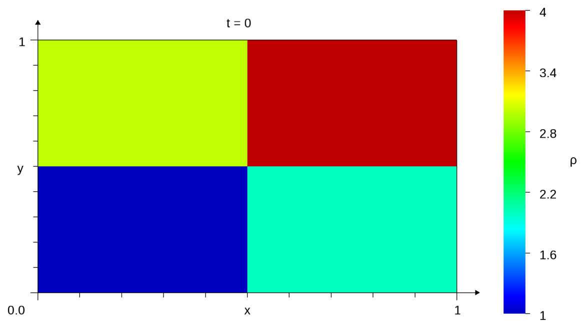
Figures 10 and 11 show cell plots of the simulation results from the 2D Riemann problem test case. The time instant is shown at the top, and the size and symbol of the plotted variable is shown by the colorbar. Figure 10a shows the initial condition (34) for the density. Figure 10b shows the final density solution at $t = 0.2$, computed by the AMR-solver. In this density field we can identify several different waves. We see the vertical shock at $x \approx 0.23$ and the horizontal shock at $y \approx 0.2$. The contact discontinuities at $x \approx 0.48$ and $y \approx 0.43$ are not as distinct, especially near the center of the domain. At $x \approx 0.7$ and $y \approx 0.7$ we find the rarefaction waves, which are wider than the other waves. We can also see that the flow variables are constant in the corners, and that we have 1D Riemann problems along the boundaries. In the central part of the domain we see complex interactions between the different waves in the form of non-linear wave fronts. Figure 10c shows that these different waves are all detected, because their surrounding areas have been refined. The contact discontinuities are not as highly refined as the shocks and rarefaction waves. Figure 10d shows the solution from the standard solver, which looks very similar to the AMR solution in Figure 10b.

Figure 11 shows the AMR solutions of the velocity components and pressure. We can see that none of these plots show all the waves. Conversely, there is no wave front that we find in Figure 11, that we do not find in Figure 10. This demonstrates that the density is a very good indicator to use in the criterion for *where* to refine.

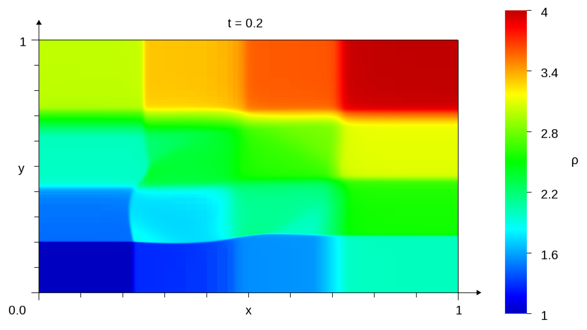
Finally for this test case, we will examine some technical output from the AMR-solver, and compare some of it with data from the structured solver. Table 7 shows selected values from the reports that the two solver produced when simulating the 2D Riemann problem.

The runtime is the measured time that the CPU used to execute the simulation. We see that the structured solver used about twice as much time as the AMR-solver. This ratio is lower than than the values we found for the shock reflection case, indicating that the efficiency gain is a little lower for this case.

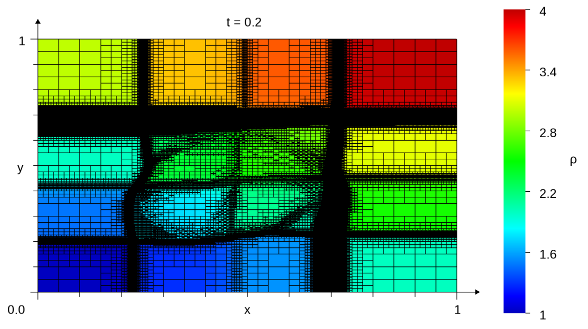
19% of the total 92.6 seconds were consumed by mesh adaptation in the AMR simulation. The remaining 81% is used for computing and transferring fluxes, computing primitive variables, etc, which is definitely the main time consumer.



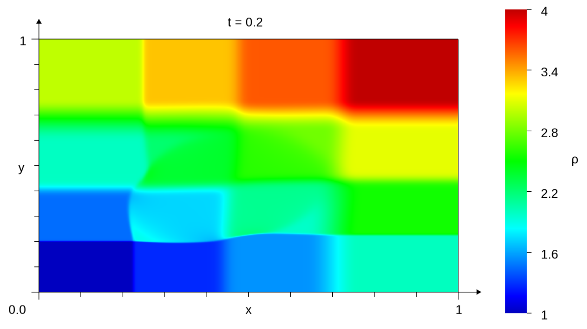
(a) Initial condition.



(b) Final solution by the AMR solver.

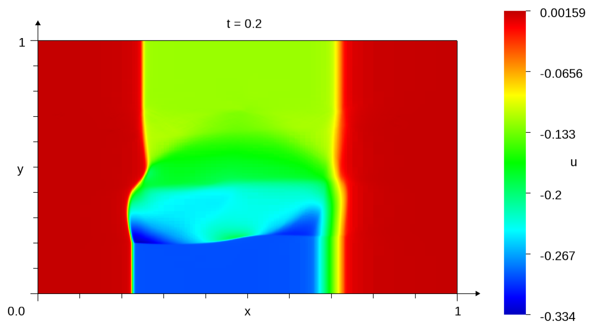


(c) AMR solution, with the mesh faces plotted as black lines.

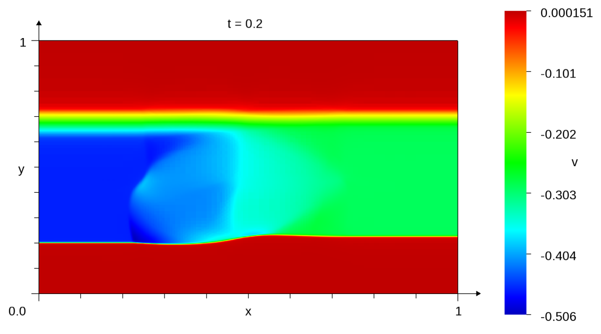


(d) Final solution by the standard solver. The uniform grid spacing Δx_{str} is equal to the smallest allowed cell size Δx_{min} for the AMR solver, cf. Table 6.

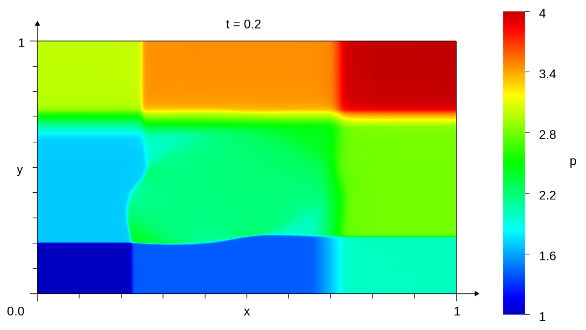
Figure 10: Cell plots of the density in the 2D Riemann problem.



(a) Velocity component u.



(b) Velocity component v.



(c) Pressure.

Figure 11: Cell plots of the velocity components and pressure in the 2D Riemann problem. These are solutions by the AMR solver.

Table 7: Miscellaneous technical output from the AMR solver. Values from the structured solver are given where applicable, for comparison. The right column is the ratio between the left and middle columns. The meaning of each row is explained when commented in the text.

	AMR	Structured	$\frac{Structured}{AMR}$
Total Runtime [s]:	92.6	187	2.02
Numerical method time [s]:	75.0	(81% of 92.6 s)	
Mesh adaptation time [s]:	17.6	(19% of 92.6 s)	
Average no. of cells:	200 321	1 638 400	8.18
Final no. of cells:	206 677	1 638 400	7.93
No. of time steps:	891	891	1
No. of mesh adaptations:	133		

The time-level-averaged number of cells and the final number of cells are approximately 8 times higher for the structured solver. This ratio was typically much higher for the shock reflection case, as we saw in Figure 8. The many waves that appear in the 2D Riemann problem require a larger portion of the domain to be well resolved. This decreases the gain from the AMR approach, as expected. Similar to the shock reflection case, we see much higher ratios in cell count than in runtime. This indicates that there is much overhead related to the data structure, since the actual mesh adaptation only used 19% of the CPU time. In turn this means that there is potential to increase the efficiency gain considerably, e.g. by optimizing the data structure. The number of time steps is how many times the explicit Euler method was applied to march the solution forward. The number of mesh adaptations is how many times the complete procedure of flagging, grading, refining and merging was executed. We can see that the mesh was adapted much more often for this test case

$$\frac{N_{adapt,Riemann}}{n_{max,Riemann}} = \frac{133}{891} = 0.149 \quad , \tag{36}$$

than for the first shock reflection case (cf. Table 1 in subsection 5.1)

$$\frac{N_{adapt,reflect}}{n_{max,reflect}} = \frac{42}{2248} = 0.0187 \quad , \tag{37}$$

where N_{adapt} is the number of adaptations and n_{max} is the number of time steps computed. We also see that the solvers used the same number of time steps, as opposed to the steady-state case in sections 5.1 and 5.2, which we saw in Figure 7.

6 CONCLUSIONS

An AMR finite volume solver using unstructured Cartesian grids has been developed for the 2D Euler equations of gas dynamics. The adaptation trigger, i.e., the criterion for *when* to adapt, is based on accumulating an approximated mass redistribution rate, cf. subsection 4.1. To the best of our knowledge, this is a new approach. The results indicate that this approach works well for both steady-state problems and transient simulations, though many more flow cases must be tested before we can assert this. For the transient test case of a 2D Riemann problem, and the development of a regular shock reflection, the new adaptation trigger keeps the mesh updated, avoiding that large gradients leave the highly resolved areas. If the resolution demanding features slow down or stop moving, the trigger activates more seldom. During convergence of the steady-state simulations, this is especially evident. This adaptation trigger is also very easy to implement and inexpensive in terms of CPU time.

The benefit from the AMR approach was measured by comparing the number of cells and the CPU time of simulations that give similar errors. The cases that benefit the most from AMR are the simulations where we allow the smallest cells, i.e., the simulations with the smallest errors and highest CPU times. In these cases, the number of cells for the AMR solver was only about 5% of the number of cells for the standard solver. The efficiency gain for CPU time is much lower, but still significant. The AMR solver used 36-40% of the runtime of the standard solver to give similar errors in the most beneficial cases. The difference between these percentages is most likely related to the mesh data structure. The mesh adaptation procedures only comprise 3-20% of the AMR solver's total runtime, whereas the numerical method consumes the rest of the runtime.

The criterion for *where* to refine and coarsen the mesh was varied. This proved to have a big effect when examining runtime, number of cells and error. A relaxed criterion gives lower cell counts and runtimes and higher errors than a strict criterion, as expected. The results indicate that to maximize the benefits from this AMR approach, the refine and merge tolerances should be set depending on the smallest allowed cell size. Specifically, with smaller cells the tolerances should be smaller, and with larger cells the tolerances should be larger, to give a good balance between CPU time and error.

References

[1] M. J. Berger and R. J. Leveque. "Adaptive Mesh Refinement Using Wave-Propagation Algorithms for Hyperbolic Systems". *SIAM Journal on Numerical Analysis* 35.6 (1998), pp. 2298–2316. ISSN: 0036-1429.

[2] R. Löhner. *Applied computational fluid dynamics techniques : an introduction based on finite element methods.* 2nd ed. Chichester: Wiley, 2008. ISBN: 9780470519073.

[3] G. W. Zumbusch. *Parallel multilevel methods : adaptive mesh refinement and load-balancing.* Ed. by H. G. Bock et al. 1st ed. Wiley-Teubner series, advances in numerical mathematics. Stuttgart, Germany: Teubner, 2003. ISBN: 3-322-80063-6.

[4] T. Linde T. Plewa and V. G. Weirs. "Adaptive mesh refinement, theory and applications : proceedings of the Chicago Workshop on Adaptive Mesh Refinement Methods, Sept. 3-5, 2003". Vol. 41. Lecture notes in computational science and engineering. Chicago Workshop on Adaptive Mesh Refinement Methods. Berlin: Springer, 2005. ISBN: 1-280-41225-9.

[5] O. P. Jacquotte. "Grid Optimization Methods for Quality Improvement and Adaption". N. P. Weatherill J. F. Thompson B. K. Soni. *Handbook of Grid Generation.* Boca Raton Fla: CRC Press, 1998.

[6] M. J. Aftosmis, M. J. Berger, and J. E. Melton. "Adaptive Cartesian Mesh Generation". J. F. Thompson, B. K. Soni, and N. P. Weatherill. *Handbook of Grid Generation.* Boca Raton Fla: CRC Press, 1998.

[7] Michael J. Aftosmis. *What is Cart3D?* 2018. URL: `https://www.nas.nasa.gov/publications/software/docs/cart3d/pages/cart3Dhome.html`.

[8] S. Müller. *Adaptive Multiscale Schemes for Conservation Laws.* Ed. by T. J. Barth et al. Lecture Notes in Computational Science and Engineering. Berlin, Heidelberg: Springer, 2003. ISBN: 978-3-540-44325-4.

[9] M. Paszynski, D. Pardo, and V. M. Calo. “A direct solver with reutilization of LU factorizations for h -adaptive finite element grids with point singularities”. *Computers and Mathematics with Applications* 65.8 (2013), pp. 1140–1151. ISSN: 0898-1221.

[10] F. Kristoffersen. “Adaptive Mesh Refinement for Conservation Laws”. Pre-Master Project. Norwegian University of Science and Technology, 2018.

[11] Randall J LeVeque. *Finite volume methods for hyperbolic problems.* eng. Cambridge, 2002.

[12] F. Kristoffersen. “Adaptive Mesh Refinement for the Euler Equations of Gas Dynamics”. Master thesis. Norwegian University of Science and Technology, 2019.