Technical University of Munich

School of Computation, Information and Technology - Informatics

Master's Thesis in Informatics

Efficient Simulation of Tsunamis Using a Dispersive Shallow Water Model

Effiziente Tsunamisimulation mit einem dispersiven Flachwassermodell

Author	David Jonathan Schneller
Supervisor	Prof. Dr. Michael Bader
Advisor	M. Sc. Lukas Krenz
Date of Submission	January 16, 2023

Eidesstattliche Erklärung

Ich versichere, dass ich diese Masterarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I confirm that this Masters's thesis is my own work and I have documented all sources and material used.

Place / Date / Signature

Abstract

The accurate simulation of tsunamis is generally deemed a computationally expensive task, since the multitude of effects usually requires a three-dimensional model. On the other hand, two-dimensional, depth-averaged systems like the shallow water equations can be simulated fairly quickly, but they only capture the tsunami wave itself and fail to capture any dispersivity, since it assumes only hydrostatic pressure. For a higher accuracy, we consider the H-BMSS-γ system [Escalante et. al, 2019]. It is hyperbolized by applying a Generalized Lagrange Multiplier procedure, and it contains a component for non-hydrostatic pressure. For the H-BMSS-γ system, we develop three methods to model the effect of a moving ocean floor: the first method assume an instantaneous undersea earthquake. The second method adds a non-hydrostatic pressure correction as a response to the sudden earthquake. Finally, the third method couples the ocean floor movement to the non-hydrostatic pressure itself. We simulate the H-BMSS-γ model with all three methods in sam(oa)^2. For the numerical discretization using the ADER-DG (Adaptive DERivative Discontinuous Galerkin) method, for which we also suggest an optimized implementation which utilizes the intrinsic tensor product structure to simplify the computational effort. Also, we avoid a nonlinear integration in the corrector step. Furthermore, we add support for adaptive mesh refinement. Finally, we show the high-order convergence of our model and the implementation, and we benchmark the three earthquake coupling methods which we derived. In particular, we see that all of them manage to capture dispersive effects.

Acknowledgments

I thank the Almighty God, and my parents for their continuous support not only through this thesis, but also during all of my higher education. Moreover, I would like to thank my advisors and my supervisor for their help and assistance with any questions I had while writing this thesis, and for many helpful discussions which I had with them.

Contents

1 Introduction

We are given the problem of simulating tsunamis, both efficiently and accurate. This means that we consider an ocean, and we assume that the ground under it moves due to an earthquake. With this, we are posed with two modelling problems: firstly, we need to model the interaction between the earthquake and the ocean, and secondly, the modelling of ocean response to a moving ground. Some models (e.g. [16]) resolve this problem by simulating both earthquake and ocean at the same time, using similar models. While this proves to be able to capture various wave effects, it also involves a lot of computational power for simulating the ocean. Other methods decouple the two parts: we are given the ground uplift which was recorded during the simulation of an earthquake. Using this data, the ocean is simulated using a different model and a different code.[1] For example, we can use the (depth-averaged) shallow water equations for this, as it is compared in e.g. [1]. Due to their two-dimensional nature, they are easy and quick to simulate. However, they only manage to capture the tsunami wave itself, and no dispersive effects. Thus, we need to look at more sophisticated depth-averaged models (e.g. [7, 8, 9, 28]) which do not neglect the non-hydrostatic pressure. In [7], a hyperbolic partial differential equation (PDE) which includes a component for non-hydrostatic pressure and vertical velocity was published. In [29], we called it the H-BMSS-γ model. It can be derived similarly to the shallow water equations, by depth-averaging the three-dimensional Euler equations [2], and then coupling the three-dimensional incompressibility condition to the non-hydrostatic pressure [7]. In [29], we presented a working simulation of said hyperbolic system in sam(oa)$^\mathbf{2}$ [19] which uses the Adaptive DERivative Discontinuous Galerkin (ADER-DG) method with a Finite Volume limiter. However, it did not include support for responding to earthquakes so far. Additionally, the H-BMSS-γ model requires small modifications to be able to handle a moving ocean floor, as it is done in e.g. [9]. The scope of this work is to extend the implementation presented there to be able to simulate tsunamis accurately and quickly.

Therefore, we structure this thesis as follows: after giving a review of the system used in [29], we turn to the simulation of tsunamis using the H-BMSS-γ system (Chapter 2). To do so, we consider three different models: the first model assumes that the earthquake happens instantly. Thus, we can simply re-use our existing simulation from [29], and only need updated initial conditions. Secondly, we model the behavior of the earthquake as time-dependent bathymetry. This requires us to modify the H-BMSS-γ model, as the non-hydrostatic pressure is directly influenced by the velocity of the uplift. We also compare this model to the fully-coupled model in [1]. The last model again, just like the first model, assumes that the earthquake happens in an instant. However, this time, we

[1]The effects of the ocean onto the crust of the Earth are neglected.

change the pressure during the uplift as well.

Moreover, the new models need to be simulated and discretized efficiently (Chapter 3). To this end, we use the tensor product structure between time and space, and we decompose the matrices using the Kronecker product which leads to a faster simulation. In addition, we suggest an alternative formulation of the ADER-DG corrector equation which avoids a nonlinear integral evaluation. Lastly, we consider Adaptive Mesh Refinement (AMR) in order to avoid computations, where they are unnecessary.

Next, we present our additions to sam(oa)^2 (Chapter 4), in order to accommodate the changes from the previous chapter. To accelerate the matrix computations which are necessary for the numerical discretization, we use YaTeTo [33] and libxsmm [13]. We integrate ASAGI [26] to read earthquake data from existing simulations. Additionally, we re-organize the code and the build system of the project.

Lastly, we show the results of our the model (Chapter 5). Firstly, we show that our model converges by using a manufactured solution with a solitary wave which is close to a solution of the model. Secondly, we demonstrate the AMR capabilities of our code. And thirdly, we examine the performance of our three tsunami models at some simple earthquake test cases from [1]. Finally, we look at a tsunami test case adapted from [18] which shows that our model captures dispersivity.

We close with a summary and an outlook on future work in Chapter 6.

Notation and Basic Assumptions Scalar functions, scalar constants and vectors are denoted by lower-case Latin alphabet letters. Vectors may also be marked with an arrow, i.e. $\vec{v}$. For the i-th component of a vector $\vec{v} \in \mathbf{R}^N$, we write $\vec{v}_i$. Matrices use uppercase Latin alphabet letters. By $\partial_i f$, we denote the partial derivative to the i-th component of the function f. The gradient is given as[2]

$$\nabla f = \begin{pmatrix} \partial_1 f \\ \vdots \\ \partial_N f \end{pmatrix}. \tag{1.0.1}$$

For two matrices A and B, we define the Kronecker product $(A \otimes B)$ by

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{pmatrix}, \tag{1.0.2}$$

where $A = (a_{ij}) \in \mathbb{R}^{m\times n}$.

Vector-valued functions are denoted by lowercase and uppercase Latin alphabet letters in bold font, e.g. $\mathbf{u}$, $\mathbf{f}$, or $\mathbf{S}$. For vectors and vector-valued functions $\mathbf{v}, \mathbf{w}$, the dot operator means

$$\mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{N} \mathbf{v}_i \mathbf{w}_i. \tag{1.0.3}$$

[2]Note that throughout this work, we only use the Cartesian coordinate system.

The divergence $\nabla \cdot \mathbf{v}$ is given as

$$\nabla \cdot \mathbf{v} = \sum_{i=1}^{N} \partial_i \mathbf{v}_i. \tag{1.0.4}$$

Uppercase calligraphic Latin alphabet letters denote higher-order tensor functions, unless noted otherwise. We write for a tensor function

$$\mathcal{F} = (\mathbf{f}^1, ..., \mathbf{f}^N). \tag{1.0.5}$$

The notation $\mathcal{F} \cdot n$ with a vector $n \in \mathbb{R}^N$ means

$$\mathcal{F} \cdot n = \sum_{i=1}^{N} n_i \mathbf{f}^i. \tag{1.0.6}$$

Similarly, $\nabla \cdot \mathcal{F}$ means

$$\nabla \cdot \mathcal{F} = \sum_{i=1}^{n} \partial_i \mathbf{f}^i. \tag{1.0.7}$$

For a vector-valued function, $\nabla \mathbf{f}$ is given as

$$\nabla \mathbf{f} = (\partial_1 \mathbf{f}, ..., \partial_N \mathbf{f}). \tag{1.0.8}$$

If $\mathbf{f}$ and $\mathbf{g}$ are two vector-valued functions, then define

$$\mathbf{f} \otimes \mathbf{g} = (\mathbf{f}_1 \mathbf{g}, ..., \mathbf{f}_N \mathbf{g}). \tag{1.0.9}$$

In this particular case, $\otimes$ does not denote the Kronecker product. Subsequently, we have

$$\nabla \cdot (\mathbf{f} \otimes \mathbf{g}) = \sum_{i=1}^{n} \partial_i (\mathbf{f}_i \mathbf{g}). \tag{1.0.10}$$

Likewise, we have bold-faced calligraphic letters which denote tensor functions of matrices. Write

$$\boldsymbol{\mathcal{B}} = (\mathbf{B}^1, ..., \mathbf{B}^N). \tag{1.0.11}$$

The dot product $\boldsymbol{\mathcal{B}} \cdot n$ is defined as for $\mathbf{F}$. In particular, for $\mathbf{f}$ differentiable, we have

$$\boldsymbol{\mathcal{B}} \cdot \nabla \mathbf{f} = \sum_{i=1}^{n} \boldsymbol{\mathcal{B}}^i \partial_i \mathbf{f}. \tag{1.0.12}$$

Furthermore, given a function $\phi : [0,1] \to \mathbf{R}^N$ which is continuously differentiable, its path integral for $f : \mathbb{R}^N \to \mathbb{R}^N$ is defined as

$$\int_\phi f(z) \, \mathrm{d}z = \int_0^1 f(\phi(t)) \phi'(t) \, \mathrm{d}t. \tag{1.0.13}$$

2 Tsunami Modelling

Most generally speaking, we consider the following situation: we have the crust of the Earth and the ocean as two separate systems which share the ocean floor as a common surface. The effects from the ocean onto the crust are neglected. Therefore, we can decouple the simulation by treating the effect from the crust onto the ocean over a change in the bathymetry. In particular, we can simulate the earthquake in advance (using e.g. a different simulator), and compute the ocean response afterwards. For the water, we may then use a cheaper depth-averaged formulation instead of a three-dimensional simulation.

2.1 Tsunami Waves

To begin with, we briefly summarize the different types of waves that we want to simulate. For a more extensive review, we refer to [28]. Roughly speaking, we differentiate between two types of waves.

The permanent change in bathymetry which occurs during the earthquake causes the main *tsunami wave*. This bathymetry change then causes the water above it to be displaced. The wave is also visible when considering the ocean to be incompressible. Since the wavelength of the tsunamis is comparably long, we have that the wave of a near-instantaneous earthquake travels at the speed $\sqrt{gh_0}$, where h_0 is a typical water height at the earthquake location. We should note that there are usually also acoustic waves propagating in the Earth crust which cause a temporary bathymetry change: we can assume that they have already left the domain as soon as we reach the end of the simulation, and they are not the target of our simulations.

The second type of waves are acoustic waves in the ocean. These are caused by the spontaneous compression of the water near the fault when the ocean floor (i.e. the bathymetry) moves, and they travel at the speed of sound, and therefore faster than the main tsunami wave.

The third type of waves are caused by dispersive effects. They move slower than the main tsunami wave, since the movement of water caused by the tsunami wave causes small changes in the density (and thus in pressure) which are cause smaller trailing waves to appear.

2.2 Depth-Averaged Equation Systems

Next, we give a brief overview over the equation system that we are going to use. Note that in this section, we assume that the bathymetry b is not time-dependent, i.e. we write $b(x, y)$.

A well-known depth-averaged system are the shallow water equations. They read

$$\begin{aligned} \partial_t h + \nabla \cdot (h\mathbf{u}) &= 0, && (2.2.1a) \\ \partial_t (h\mathbf{u}) + \nabla \cdot (h\mathbf{u} \otimes \mathbf{u}) + \frac{g}{2}\nabla(h^2) &= -gh\nabla b. && (2.2.1b) \end{aligned}$$

Here, $h(t, x, y)$ is the water height, and $\mathbf{u}(t, x, y) = (u(t, x, y), v(t, x, y))$ is the horizontal velocity vector in two dimensions.[1] We write for the water height $\eta = h + b$. The constant g denotes the gravitational acceleration. Naturally, we require $h \geq 0$. All in all, the equation system (2.2.1) is strictly hyperbolic for $h > 0$.

The first equation (2.2.1a) is called the mass conservation equation, and the second equation (2.2.1b) is called the momentum conservation equation.[2]

2.2.1 The Governing Equations

The system which we are going to mainly consider was introduced in [7], and it is named H-BMSS-γ in [29]. It reads

$$\begin{aligned} \partial_t h + \nabla \cdot (h\mathbf{u}) &= 0, && (2.2.2a) \\ \partial_t (h\mathbf{u}) + \nabla \cdot (h\mathbf{u} \otimes \mathbf{u}) + \nabla \left(\frac{g}{2}h^2 + hp\right) &= -(gh + \gamma p)\nabla b, && (2.2.2b) \\ \partial_t (hw) + \nabla \cdot (h\mathbf{u}w) &= \gamma p, && (2.2.2c) \\ \partial_t (hp) + \nabla \cdot (h\mathbf{u}(p + c^2)) - c^2(\mathbf{u} \cdot \nabla)h &= 2c^2(\mathbf{u} \cdot \nabla)b - 2c^2 w. && (2.2.2d) \end{aligned}$$

Once again, $h(t, x, y)$ denotes the water height and $\mathbf{u}(t, x, y)$ the horizontal velocity. In addition, $w(t, x, y)$ is the vertical velocity, $p(t, x, y)$ is the non-hydrostatic pressure and $b(x, y)$ is the bathymetry. For the constants, we have g as the gravitational acceleration, γ as model parameter, and c as an artificial pressure wave phase speed.

In short, the system can be understood as extending the shallow water equations with a dynamic pressure component which is coupled to a divergence-free condition using the parameter c. The equations (2.2.2c) and (2.2.2d) form a subsystem resembling the wave equation which is coupled with the subsystem consisting of (2.2.2a) and (2.2.2b) which is similar to the shallow water equations. For more details on the derivation of (2.2.2), see [2, 7, 29].

When ignoring the source terms, the system is hyperbolic, given that the following two positivity properties

$$\begin{aligned} h &> 0, && (2.2.3a) \\ C(h, p) = gh + p + c^2 &> 0 && (2.2.3b) \end{aligned}$$

are fulfilled (cf. [29]). With flat bathymetry (i.e. $\nabla \cdot b = 0$), we also have strict hyperbolicity, if (2.2.3) are given.

[1] The equation system can be freely extended to higher or lower-dimensional cases—but for the scope of this work, we will always use two dimensions.

[2] Some variants, e.g. in [7] add a friction term to the momentum equation.

If we set $c = 0$ and $\gamma = 0$ in (2.2.2), we retrieve the shallow water equations (2.2.1).

In [7] and [29], an optional friction term is added to (2.2.2b), and a wave-breaking term is added to (2.2.2c). But since we do not use them in this work, we omitted them in (2.2.2).

2.2.2 Eigenstructure

We can write (2.2.2) in a more compact formulation (as it is done in [29]) which is

$$\partial_t \mathbf{q} + \nabla \cdot \mathcal{F}(\mathbf{q}) + \boldsymbol{\mathcal{B}}(\mathbf{q}) \cdot \nabla \mathbf{q} = \mathbf{S}(\mathbf{q}, \nabla \mathbf{q}). \tag{2.2.4}$$

Here, we have

$$\mathbf{q}(t,x,y) = \begin{pmatrix} h \\ hu \\ hv \\ hw \\ hp \\ b \end{pmatrix}. \tag{2.2.5}$$

In addition, we write

$$\mathbf{f}^1(q) = \begin{pmatrix} hu \\ h(u^2+p) \\ huv \\ huw \\ hu(c^2+p) \\ 0 \end{pmatrix}, \ \mathbf{f}^2(q) = \begin{pmatrix} hv \\ hvu \\ h(v^2+p) \\ hvw \\ hv(c^2+p) \\ 0 \end{pmatrix}, \tag{2.2.6}$$

and

$$\mathbf{B}^1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ gh & 0 & 0 & 0 & 0 & (gh+\gamma p) \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ c^2u & 0 & 0 & 0 & 0 & 2c^2u \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \ \mathbf{B}^2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ gh & 0 & 0 & 0 & 0 & (gh+\gamma p) \\ 0 & 0 & 0 & 0 & 0 & 0 \\ c^2v & 0 & 0 & 0 & 0 & 2c^2v \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{2.2.7}$$

Then, we can define

$$\mathcal{F} = (\mathbf{f}^1, \mathbf{f}^2), \tag{2.2.8}$$

and

$$\boldsymbol{\mathcal{B}} = (\mathbf{B}^1, \mathbf{B}^2). \tag{2.2.9}$$

Moreover, we define

$$\boldsymbol{\mathcal{A}} = D\mathcal{F} + \boldsymbol{\mathcal{B}}. \tag{2.2.10}$$

Here, $D\mathcal{F}$ means

$$D\mathcal{F} = (D\mathbf{f}^1, D\mathbf{f}^2), \tag{2.2.11}$$

where $D\mathbf{f}^k$ denotes the Jacobian of $\mathbf{f}^k$ for $k = 1, 2$. Finally, the source term $\mathbf{S}$ is given by

$$\mathbf{S}(\mathbf{q}, \nabla\mathbf{q}) = \begin{pmatrix} 0 \\ 0 \\ \gamma p \\ -2c^2 w \\ 0 \end{pmatrix}. \tag{2.2.12}$$

Note that τ_b is vector-valued, and hence, $\mathbf{S}$ still maps to a vector of dimension 6. By using $\boldsymbol{\mathcal{A}}$, we can compactify the notation even more to

$$\partial_t \mathbf{q} + \boldsymbol{\mathcal{A}}(\mathbf{q}) \cdot \nabla\mathbf{q} = \mathbf{S}(\mathbf{q}, \nabla\mathbf{q}). \tag{2.2.13}$$

With this notation, we are going to shortly give the eigenstructure of (2.2.2) as described in [29] (and incorrectly described in [7]). In particular, we look at $\mathbf{A}^1$, where

$$\boldsymbol{\mathcal{A}} = (\mathbf{A}^1, \mathbf{A}^2). \tag{2.2.14}$$

Note that $\mathbf{A}^1$ includes the bathymetry as a component. The eigenvalues and eigenvectors are given as

$$\begin{aligned}
\lambda_1 &= u - \sqrt{C}, & r_1 &= (1, u - \sqrt{C}, v, w, p + c^2, 0)^T, && \text{(2.2.15a)}\\
\lambda_2 &= u, & r_2 &= (1, u, 0, 0, -gh, 0)^T, && \text{(2.2.15b)}\\
\lambda_3 &= u + \sqrt{C}, & r_3 &= (1, u + \sqrt{C}, v, w, p + c^2, 0)^T, && \text{(2.2.15c)}\\
\lambda_v &= u, & r_v &= (0, 0, 1, 0, 0, 0)^T, && \text{(2.2.15d)}\\
\lambda_w &= u, & r_w &= (0, 0, 0, 1, 0, 0)^T, && \text{(2.2.15e)}\\
& & & && \text{(2.2.15f)}
\end{aligned}$$

and for the bathymetry:

$$\lambda_b = 0,\ r_b = \begin{pmatrix} 2c^2 + gh + \gamma p \\ 0 \\ \left(2c^2 + gh + \gamma p\right) v \\ \left(2c^2 + gh + \gamma p\right) w \\ 2c^2u^2 + gh(p - c^2) + \gamma p(p + c^2) \\ u^2 - C \end{pmatrix}. \tag{2.2.15g}$$

2.2.3 Solitary Waves

Next, we are going to give a short overview over a special case of (2.2.2) and its solution.

For the solitary wave, we are given a water height $H > 0$ and a wave amplitude $A > 0$. In addition, we will only consider the case $c \to \infty$. Then, we employ a change of variables using

$$\xi = \frac{x - c_A t}{l}. \tag{2.2.16}$$

Method	Description
1a	Instant uplift
1b	Instant uplift with pressure correction
2	Time-dependent bathymetry

Table 2.1: The different tsunami modelling methods for (2.2.2).

Here, the constants c_A and l are defined as

$$c_A = \sqrt{g(A+H)}, \tag{2.2.17}$$

$$l = H\sqrt{\frac{2}{\gamma A}(A+H)}, \tag{2.2.18}$$

respectively. In particular, c_A is different from c.

For $c \to \infty$, we obtain the solution

$$h = H + A(\operatorname{sech}(\xi))^2, \tag{2.2.19a}$$

$$u = c_A \frac{h-H}{h}, \tag{2.2.19b}$$

$$w = -\frac{Ac_A H}{lh} \operatorname{sech}(\xi) \operatorname{sech}'(\xi), \tag{2.2.19c}$$

$$p = \frac{Ac_A^2 H^2}{2l^2h^2} \Big((2H-h) \left(\operatorname{sech}'(\xi)\right)^2 + h\operatorname{sech}(\xi) \operatorname{sech}''(\xi) \Big). \tag{2.2.19d}$$

Here sech denotes the secans hyperbolicus given by

$$\operatorname{sech}(x) = \frac{2}{e^x + e^{-x}}. \tag{2.2.20}$$

The system (2.2.19) is also described in [2]. For finite values of c, a quasi-exact solution is described in [29] (which corrects some incorrect formulas from [7]).

2.3 Modelling the Bathymetry Influence on the Governing Equations

With the previously-described equation system, we now want to simulate the tsunami waves described in Section 2.1. That is, now we look at a time-dependent bathymetry $b(t, x, y)$. For our purposes, we consider the situation where an earthquake occurs in the timeframe $[0, t^{\text{eq}})$ for some $t^{\text{eq}} \geq 0$, and that the bathymetry does not change afterwards: $\partial_t b(t, \cdot\,, \cdot\,) \equiv 0$ for $t > t^{\text{eq}}$.

All methods which we derived are listed in Table 2.1. They mainly form the analogues to the treatment of the shallow water equations (2.2.1), as discussed in [1, Methods 2 and 3].

2.3.1 Immediate Earthquake (Method 1a)

The simplest method is to assume that the earthquake happens in an instant, i.e. we take the bathymetry

$$b_{\lim}(x,y) = \lim_{t\to\infty} b(t,x,y) = b(t^{\mathrm{eq}},x,y). \tag{2.3.1}$$

Then, we insert $b_{\lim}$ into (2.2.2), while all other variables and initial conditions stay the same. In particular, this means that h stays as it was before, but $\eta = h + b_{\lim}$ uses the limit bathymetry. This method can be run with the existing simulation code from [29] immediately, as we merely change the initial conditions.

In [1], the same method is tested for the shallow water equations, and it shows to work well for small t^{eq}. Naturally, the shallow water equations only capture the tsunami wave; therefore, we expect the method applied to (2.2.2) to not only show the tsunami wave, but also dispersion effects.

2.3.2 Time-Dependent Bathymetry (Method 2)

Other than adjusting the time-dependent bathymetry to our existing equations, we can do the opposite and adjust our equation system to support time-dependent bathymetry. This was also done for similar systems like [9].

Extending the Governing Equations to Time-Dependent Bathymetry

As indicated by [2, Remark 3], we merely need to modify equation (2.2.2d). The equations (2.2.2a), (2.2.2b), and (2.2.2c) do not change their form when we have time-dependent bathymetry.[3] Thus, we look at the original equation where (2.2.2d) is derived from. It is

$$\partial_t(\eta^2 - b^2) + \nabla \cdot \big((\eta^2 - b^2)\mathbf{u}\big) = 2hw \tag{2.3.2}$$

which is given in the one-dimensional form in [2, eq. (40)]. Using that $\eta = h + b$, we obtain

$$\eta^2 - b^2 = h(h + 2b) \tag{2.3.3}$$

Together with the chain rule, this yields

$$(h + 2b)\left(\partial_t h + \nabla \cdot (h\mathbf{u})\right) + h\left(\partial_t(h + 2b) + (\mathbf{u}\cdot\nabla)(h + 2b)\right) = 2hw. \tag{2.3.4}$$

Inserting the mass conservation equation (2.2.2a) to remove the first term as well as to replace $\partial_t h$, we obtain

$$0 + h\left(-\nabla \cdot (h\mathbf{u}) + 2\partial_t b + (\mathbf{u}\cdot\nabla)(h + 2b)\right) = 2hw. \tag{2.3.5}$$

Next, we divide by h to obtain

$$-\nabla \cdot (h\mathbf{u}) + (\mathbf{u}\cdot\nabla)(h + 2b) = 2w - 2\partial_t b. \tag{2.3.6}$$

[3] Likewise, the shallow water equations also do not gain any new terms.

Lastly, we once again apply the GLM procedure again (see e.g. [29]) and couple (2.3.6) with the continuity equation for hp. This yields

$$\partial_t(hp) + \nabla \cdot (h\mathbf{u}(p + c^2)) - c^2(\mathbf{u} \cdot \nabla)h = 2c^2((\mathbf{u} \cdot \nabla)b + \partial_t b - w). \tag{2.3.7}$$

Thus, in comparison to the non-time dependent method, we only gained the term $2c^2\partial_t b$ on the right-hand side as another source term.

The Resulting System

In total, the new system reads

$$\begin{aligned}
\partial_t h + \nabla \cdot (h\mathbf{u}) &= 0, && (2.3.8a)\\
\partial_t(h\mathbf{u}) + \nabla \cdot (h\mathbf{u} \otimes \mathbf{u}) + \nabla\left(\frac{g}{2}h^2 + hp\right) &= -(gh + \gamma p)\nabla b, && (2.3.8b)\\
\partial_t(hw) + \nabla \cdot (h\mathbf{u}w) &= \gamma p, && (2.3.8c)\\
\partial_t(hp) + \nabla \cdot (h\mathbf{u}(p + c^2)) - c^2(\mathbf{u} \cdot \nabla)h &= 2c^2((\mathbf{u} \cdot \nabla)b + \partial_t b - w). && (2.3.8d)
\end{aligned}$$

These changes in the equations conform with similar models from e.g. [8, 9]. In these models, the effects of time-dependent bathymetry are included in the same way as done in (2.3.8), as far as comparable.[4] The counteraction of $\partial_t b$ to w has an intuitive explaination as follows (assuming a flat bathymetry): an uplift of the seafloor causes the ocean water to be effectively compressed. In our equation system, this translates to an increase of pressure. In contrast, a positive value for w means that the water wants to expand, and therefore we have a decrease in pressure.

Once again, a comparable method is used for the shallow water equations in [1]. Just as in the instantaneous case, we can expect a similar behavior for (2.3.8), but with additional dispersion effects.

Model Test Cases

In order to investigate the changed equations more closely, we look at a simple example. As initial conditions, we assume a flat bathymetry, as well as a constant water height and no velocity. We also assume w and p to be constant in space. Thus, we have

$$h(0, \cdot\,) \equiv H, \tag{2.3.9}$$

$$\mathbf{u}(0, \cdot\,) \equiv 0, \tag{2.3.10}$$

$$w(0, \cdot\,) \equiv w_0, \tag{2.3.11}$$

$$p(0, \cdot\,) \equiv p_0. \tag{2.3.12}$$

Here $w_0, p_0, H \in \mathbb{R}$ are three constants, and we require $H > 0$. The bathymetry is described by a function which depends only on the time, i.e. there is a differentiable function f, so that

$$b(t, \cdot\,) \equiv f(t) \tag{2.3.13}$$

[4]This means: looking at the equations for the non-hydrostatic pressure hp near the bathymetry, and look to the contribution of the vertical velocity w.

for all t. As a result, $\nabla b \equiv 0$. All other constants can be chosen freely. Inserting the conditions into (2.3.8) yields that h and $\mathbf{u}$ stay constant in time, since all spatial derivatives are zero. Thus, we are ultimately left with the system

$$\partial_t w = \frac{\gamma}{H} p, \tag{2.3.14a}$$

$$\partial_t p = \frac{2c^2}{H}(\partial_t b - w) \tag{2.3.14b}$$

which can be seen as an inhomogenous ordinary differential equation (ODE). In short, we could imagine this scenario to happen in a zero-dimensional domain which only consists of a single point. This is because these test scenarios assume a flat water surface, and thus the pressure cannot be converted into horizontal velocity. Furthermore, the incompressibility assumption prevents any change in water height related to the pressure.

If $\gamma = 0$, then we have the immediate solution for (2.3.14) that

$$w(t) = w_0, \tag{2.3.15a}$$

$$p(t) = \frac{2c^2}{H}(b(t) - b_0 - tw_0) + p_0. \tag{2.3.15b}$$

If $\gamma > 0$, we as usually for an inhomogenous linear ODE system (see e.g. [32]). For that, write

$$\omega = c\sqrt{\frac{2\gamma}{H}}. \tag{2.3.16}$$

Then we begin to look at the homogenous equation (i.e. $\partial_t b \equiv 0$)

$$\partial_{tt} w = \gamma \partial_t p = -\omega^2 w \tag{2.3.17}$$

which has the family of solutions

$$w^H_{A,B}(t) = A\cos(\omega t) + B\sin(\omega t), \tag{2.3.18}$$

where A and B depend on p_0 and w_0. Next, we solve the inhomogenous equation

$$\partial_{tt} w = -\omega^2(w - \partial_t b). \tag{2.3.19}$$

For this, we use the "variation of constants" method [32]: for A, we insert a function $A(t)$, and for B, we insert a function $B(t)$. After differentiating $w^H_{A(t),B(t)}(t)$ by t twice and comparing to the homogenous solution, we impose the following conditions:

$$w^H_{\partial_t A(t),\partial_t B(t)}(t) = 0, \tag{2.3.20}$$

$$\partial_t w^H_{\partial_t A(t),\partial_t B(t)}(t) = \omega^2 \partial_t b. \tag{2.3.21}$$

This can be written as

$$\begin{pmatrix} \cos(\omega t) & \sin(\omega t) \\ -\omega\sin(\omega t) & \omega\cos(\omega t) \end{pmatrix} \begin{pmatrix} \partial_t A(t) \\ \partial_t B(t) \end{pmatrix} = \begin{pmatrix} 0 \\ \omega^2 \partial_t b \end{pmatrix}. \tag{2.3.22}$$

Solving this linear system yields

$$\partial_t \begin{pmatrix} A(t) \\ B(t) \end{pmatrix} = \omega \partial_t b(t) \begin{pmatrix} -\sin(\omega t) \\ \cos(\omega t) \end{pmatrix} = \omega \partial_t b(t) \begin{pmatrix} \sin(-\omega t) \\ \cos(-\omega t) \end{pmatrix}. \tag{2.3.23}$$

Integrating over t gives us an expression for $A(t)$ and $B(t)$. Thus, by further simplifications and re-substituting into $w_{A(t),B(t)}(t)$, we obtain

$$w_{A_0,B_0}(t) = w^H_{A_0,B_0}(t) + \omega \int_0^t \partial_t b(s) \sin(\omega(t-s))\, \mathrm{d}s. \tag{2.3.24}$$

Here A_0, B_0 are determined from p_0 and w_0. In addition, we obtain

$$\partial_t w_{A_0,B_0}(t) = \partial_t w^H_{A_0,B_0}(t) + \omega^2 \int_0^t \partial_t b(s) \cos(\omega(t-s))\, \mathrm{d}s. \tag{2.3.25}$$

The integral derivative vanishes here, since we have a sine under the integral. Furthermore, we have by differentiating again and recalling that $\partial_{tt} w^H_{A_0,B_0}(t) = -\omega^2 w^H_{A_0,B_0}(t)$ that

$$\begin{aligned} &\partial_{tt} w_{A_0,B_0}(t) \\ &= \partial_{tt} w^H_{A_0,B_0}(t) - \omega^2 \left(\partial_t b(t) + \omega \int_0^t \partial_t b(s) \sin(\omega(t-s))\, \mathrm{d}s \right) \\ &= -\omega^2 (w_{A_0,B_0}(t) - \partial_t b). \end{aligned} \tag{2.3.26}$$

The term $\partial_t b(t)$ stems from the integral derivative. In total, this shows that our choice for w_{A_0,B_0} is a solution to (2.3.14). We also obtain

$$\begin{aligned} &p_{A_0,B_0}(t) \\ &= \frac{1}{\gamma} \partial_t w_{A_0,B_0}(t) \\ &= \frac{1}{\gamma} \partial_t w^H_{A_0,B_0}(t) + \frac{2c^2}{H} \int_0^t \partial_t b(s) \cos(\omega(t-s))\, \mathrm{d}t. \end{aligned} \tag{2.3.27}$$

If we consider starting with a resting lake, i.e. $p_0 = 0$ and $w_0 = 0$, then we only need to specify b, or rather, $\partial_t b$. For example, suppose that for some $C \in \mathbb{R}$, we have

$$\partial_t b \equiv C. \tag{2.3.28}$$

Then, we obtain

$$p(t) = \frac{2c^2 C}{H} \int_0^t (\cos(\omega(t-s)))\, \mathrm{d}s = Cc\sqrt{\frac{2}{\gamma H}} \sin(\omega t). \tag{2.3.29}$$

And we also get w as[5]

$$w(t) = C\omega \int_0^t \sin(\omega(t-s))\, \mathrm{d}s = C(1 - \cos(\omega t)) = 2C \left(\sin\left(\frac{\omega}{2} t \right) \right)^2. \tag{2.3.30}$$

[5] The identity $1 - \cos(2x) = 2\sin^2(x)$ can be seen by subtracting the identities $1 = \cos^2(x) + \sin^2(x)$ and $\cos(2x) = \cos^2(x) - \sin^2(x)$.

2.3.3 Pressure-Corrected Instantaneous Source Method (Method 1b)

We revisit the instantaneous source method: suppose that once again, t^{eq} is very small, so that we can ignore all spatial derivatives. This puts us into the situation that we discussed in the previous section, i.e. h, u, and v do not change. This time, we additionally assume that due to t^{eq} being small, p and w do not influence each other.[6] Thus, we are left with

$$\partial_t(hw) = 0, \tag{2.3.31}$$

$$\partial_t(hp) = 2c^2\partial_t b. \tag{2.3.32}$$

Integrating both equations yields

$$(hw)(t^{\mathrm{eq}}) = w_0, \tag{2.3.33}$$

$$(hp)(t^{\mathrm{eq}}) = p_0 + 2c^2(b(t^{\mathrm{eq}}) - b(0)). \tag{2.3.34}$$

We are also able to deduce this result from (2.3.24) and (2.3.27) derived in the previous section, without looking at (2.3.31). For that, we assume that we have t^{eq} but very small, so that we can ignore all terms which are polynomial in t^{eq}, but $b(t^{\mathrm{eq}})$ is not set to $b(0)$. Then, given $k \geq 0$, we have that for t^{eq} close to 0 that

$$\left|\int_0^{t^{\mathrm{eq}}} t^k \partial_t b(s)\,\mathrm{d}s\right| \approx \begin{cases} |b(t^{\mathrm{eq}}) - b(0)| & k = 0, \\ 0 & k > 0. \end{cases} \tag{2.3.35}$$

This is because we can bound

$$\begin{aligned} &\left|\int_0^{t^{\mathrm{eq}}} t^k \partial_t b(s)\,\mathrm{d}t\right| \\ &\leq \left|\int_0^{t^{\mathrm{eq}}} \partial_t b(s)\,\mathrm{d}s\right| \left(\max_{s\in[0,t^{\mathrm{eq}}]} s^k\right) \\ &= |b(t^{\mathrm{eq}}) - b(0)|\,(t^{\mathrm{eq}})^k. \end{aligned} \tag{2.3.36}$$

Since $|b(t^{\mathrm{eq}}) - b(0)|$ is constant, we have that the whole term can be seen as approximatively 0 for t^{eq} close to 0, if $k > 0$. Using this, we obtain for hp that

$$(hp)(t^{\mathrm{eq}}) = 2c^2 \int_0^{t^{\mathrm{eq}}} \partial_t b(s) \cos\left(\omega(s - t^{\mathrm{eq}})\right) \mathrm{d}s \approx -2c^2(b(t^{\mathrm{eq}}) - b(0)), \tag{2.3.37}$$

due to $\cos(x) = 1 + o(x)$ for $x \to 0$. For w, this gives us

$$(hw)(t^{\mathrm{eq}}) = \omega \int_0^{t^{\mathrm{eq}}} \partial_t b(s) \sin\left(\omega(s - t^{\mathrm{eq}})\right) \mathrm{d}s \approx 0, \tag{2.3.38}$$

since $\sin(x) = o(x)$ for $x \to 0$. Thus, the whole term can be neglected for t^{eq} close to 0.

If we consider the shallow water case, method 1b collapses into method 1a. The pressure correction can be understood as follows: the sudden seafloor movement which happens without spatial movement should effectively cause a sudden compression of the surrounding water—thus it translates into an increase in pressure.

[6] This is equivalent to $\gamma = 0$.

3 Numerical Updates

We next present the improvements to our numerical scheme as compared to [29]. This constitutes mostly two main points: the improvement in the computation of the ADER-DG corrector step, and the utilization of the tensor product structure in space and time.

3.1 The ADER-DG Method

The method which we use to discretize and simulate the equations is called the Adaptive DERivative Discontinuous Galerikin, or abbreviated, ADER-DG method. Subdividing a space into cells, we choose a function space in both space and time.

1. *Prediction Step:* we compute a local time evolution on each cell, for a given function space in space and time. Thus, we usually end up searching for a fixed point per cell, and we do not communicate with any neighboring element.

2. *Correction Step:* given the time evolution from the prediction step, we compute the result for the next timestep. Here, we also take the interaction with the neighboring cells into account, effectively amounting to the evaluation of the numerical fluxes at the boundary. Thus, we are required to exchange information with the directly adjacent cells.

These steps are repeated until we reach the target simulation time. Due to Godunov's Theorem[12], we are naturally expected to run into spurious oscillations near shocks or contact disconinuities. The usual strategy is to employ a limiter (see e.g. [34]) in this situation which is triggered when the method described above runs into problems (e.g. the fixed-point iteration diverges). In addition, heuristic strategies like the "Discrete Maximum Principle"[6, 34] can be employed.

For a more extensive introduction to the background of ADER-DG, see [29] or one of the original papers (e.g. [4, 34]). We are going to discuss some parts of the method in the following, since we have improved our implementation compared to [29].

The finite volume method is used for the limiter in our code. It assumes piece-wise constant functions, and we only exchange numerical fluxes between neighboring cells. The method equals an ADER-DG method when using a function space which only consists of constant functions in space and time: then the prediction step is skipped completely, and in the correction step only adds the numerical fluxes from the neighboring cells. Other than that, source terms can be integrated using a splitting scheme.

3.2 Preliminaries

For discussing the ADER-DG method, we need two function spaces: one function space S_C denoting the corrector function space, and a function space S_P denoting the function space for the predictor. Both S_C and S_P are assumed to be finite-dimensional. Note that S_C and S_P contain vector-valued functions: functions in S_C map $\mathbb{R}^2 \to \mathbb{R}^6$, and functions in S_P map $\mathbb{R}^3 \to \mathbb{R}^6$. The target space is the space of conserved variables. By N_P, we denote the polynomial degree of S_C and S_P.

In this work, we are going to assume that S_P has a tensor product structure, i.e. it can be decomposed into

$$S_P = S_T \otimes S_C := \{f(t,\vec{x}) \mid f(t,\vec{x}) = \phi(t) \odot \psi(\vec{x});\ \phi \in S_T,\ \psi \in S_C\}, \tag{3.2.1}$$

where S_T is a function space in time. The operator $\odot$ denotes the point-wise vector multiplication.

Moreover, we assume that for each of the vector components, S_C and S_P have the same basis.[1] We assume that we have given a basis of *scalar* functions written as B_Z for S_Z, where $Z = T, C, P$. We write

$$B_C = \{v_1, ..., v_{D_C}\}, \tag{3.2.2}$$
$$B_P = \{w_1, ..., w_{D_P}\}, \tag{3.2.3}$$
$$B_T = \{z_1, ..., z_{D_T}\}, \tag{3.2.4}$$

we can decompose each w_k into v_i and z_j, where $k = jD_C + i$, so that

$$w_k(t,\vec{x}) = z_j(t)v_i(\vec{x}). \tag{3.2.5}$$

Secondly, we look at the numerical fluxes. Firstly, we introduce the notion of a family of paths by $\Phi[q^-, q^+] : [0,1] \to \mathbb{R}^6$. Here q^- and q^+ are given end points. We require the family of paths to fulfill for all q^- and q^+ (when chosen appropriately) that

$$\Phi[q^-, q^+](0) = q^-, \tag{3.2.6}$$
$$\Phi[q^-, q^+](1) = q^+, \tag{3.2.7}$$
$$\Phi[q^-, q^-] \equiv q^-. \tag{3.2.8}$$

The first two conditions determine the end point, and the last condition states that if start and end point are the same, then the whole path needs to be constant. Moreover, we require $(q^-, q^+, t) \mapsto \Phi[q^-, q^+](t)$ to be locally Lipschitz continuous in all three variables.[2] A more formal introduction to the topic is given in [29] or [23].[3] Given a family

[1]This means that we have the same basis for the variable h, hu, hv, hw, hp, and b.

[2]In the case of the third variable t, this results in Lipschitz continuity on all of $[0,1]$.

[3]When considering the Riemann problem, the whole concept of a family of paths is mainly needed for quantifying the shocks in the non-conservative case, i.e. concerning the matrix $\boldsymbol{\mathcal{B}}$. In the conservative case, the family of paths is not needed, and in the case of a contact discontinuity or rarefaction, we can deduce a suitable choice of paths by an associated ODE from an eigenvector of $\boldsymbol{\mathcal{A}}$. For details, see e.g. [23].

of paths, we may introduce a prototypical numerical flux given by

$$\mathcal{N}(q^-,q^+) = \frac{1}{2}\left(\mathcal{F}(q^+) + \mathcal{F}(q^-) + \int_{\Phi[q^-,q^+]} \boldsymbol{\mathcal{B}}(q)\,\mathrm{d}q - \int_{\Phi[q^-,q^+]} \Theta[q^-,q^+](q)\,\mathrm{d}q\right). \tag{3.2.9}$$

This formulation encompasses a large family of approximate numerical fluxes, including HLL, Roe, etc. [3], or the DOT method [5].[4] It should be noted that in this structure, the numerical flux treats parts of the equation differently, depending on if they belong to the conservative part in $\mathcal{F}$ or the non-conservative part in $\boldsymbol{\mathcal{B}}$. For the scope of this work, we are going to exclusively rely on a Rusanov-type flux, i.e. $\Theta[q^-,q^+]$ reads

$$\Theta[q^-,q^+](q) = \max\left\{s(q^-), s(q^+)\right\}\begin{pmatrix}1&0&0&0&0&1\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&0\end{pmatrix}. \tag{3.2.10}$$

Here $s(q)$ is the maximum wave speed of the vector $q = (h, hu, hv, hw, hp, b)^T$. It is given by

$$s(q) = \sqrt{u^2+v^2} + \sqrt{gh+p+c^2}. \tag{3.2.11}$$

The first row of the matrix in (3.2.10) is chosen to have no influence on the water height η in a Resting Lake scenario.[5] Since $\Theta[q^-,q^+]$ is constant (it does only depend on q^- and q^+), we obtain

$$\int_{\Phi[q^-,q^+]} \Theta[q^-,q^+](q)\,\mathrm{d}q = \Theta[q^-,q^+](q^+-q^-). \tag{3.2.12}$$

For more details about the numerical fluxes and other choices we refer to [29].

In this work, we chose to replace the numerical fluxes by *numerical fluctuations* which are defined as

$$\mathcal{D}(q^-,q^+) = \mathcal{N}(q^-,q^+) - \mathcal{F}(q^-). \tag{3.2.13}$$

This yields for a normal vector n that

$$\mathcal{D}(q^-,q^+)\cdot n = \int_{\Phi[q^-,q^+]} \boldsymbol{\mathcal{A}}(u)\cdot n\,\mathrm{d}u - \int_{\Phi[q^-,q^+]} \Theta[q^-,q^+](q)\cdot n\,\mathrm{d}q, \tag{3.2.14}$$

In particular, a different treatment of the conservative and the non-conservative part is no longer needed.[6] Additionally, we have for q that

$$\mathcal{D}(q,q) = 0, \tag{3.2.15}$$

[4]In particular, PVM methods [3] have $\Theta[q^-,q^+](q)$ to be a constant function in q, i.e. only dependent on q^- and q^+. In contrast, the Θ from the DOT method [5] does depend on q, and thus the result in general also depends on the choice of paths.

[5]That is: for $q^- = (h^-,0,0,0,0,b^-)^T$ and $q^+ = (h^+,0,0,0,0,b^+)^T$, so that $h^- + b^- = h^+ + b^+$, we obtain $\Theta = 0$.

[6]It is advantageous to still know about $\mathcal{F}$ when computing the path integral, in order to reduce the computational effort: for $\mathcal{F}$ the path integral always collapses to $(\mathcal{F}(q^+) - \mathcal{F}(q^-))$.

if $\mathcal{N}$ is consistent (which is the case for (3.2.9)).

3.3 The Predictor Iteration

The first part in an ADER-DG step is the cell-wise predictor iteration. Let $\mathbf{u}^n \in S_C$ be the function representing the solution at timestep t_n. Then we are looking for a $\mathbf{q} \in S_P$, so that the equation

$$\begin{aligned}
&\int_T \mathbf{w}(t_{n+1},\vec{x})^T \mathbf{q}(t_{n+1},\vec{x})\ \mathrm{d}\vec{x} - \int_{[t_n,t_{n+1}]\times T} \partial_t \mathbf{w}(t,\vec{x})^T \mathbf{q}(t,\vec{x})\ \mathrm{d}(t,\vec{x}) \\
&= \int_T \mathbf{w}(t_n,\vec{x})^T \mathbf{u}^n(\vec{x})\ \mathrm{d}\vec{x} \qquad (3.3.1)\\
&- \int_{[t_n,t_{n+1}]\times T} \mathbf{w}(t,\vec{x})^T \left(\boldsymbol{\mathcal{A}}(\mathbf{q}) \cdot \nabla \mathbf{q} - \mathbf{S}(\mathbf{q}, \nabla \mathbf{q})\right) \mathrm{d}(t,\vec{x})
\end{aligned}$$

is fulfilled for all $\mathbf{w} \in S_P$. It shall be noted that the vector-valued test functions are used slightly differently in comparison to [29] and [7], but the equations can be shown to be equivalent nonetheless. However, for equivalence to the formulation from [7], we need to use the same function space in each component of the vector-valued functions.

The predictor is derived by testing the original PDE and integrating it in space and time. After that, we use integration by parts in time, and replace $q(t_n, \vec{x})$ by $\mathbf{u}^n$ (this happens in the second row in (3.3.1)). For more details, we refer to [29]. Most notably, the underlying PDE from (2.2.13) is not altered in (3.3.1), except for the time derivative. In fact, the predictor iteration as in (3.3.1) can be thought of as applying the method of lines: we test and integrate the original equation in space only which yields an ODE per test function. For the time integration, we can now choose any ODE integration method that looks suitable to us.[7] On the other hand, we see by this as well that the ADER-DG predictor includes an ODE integrator[8] which can be seen as a weak form of the Power Series Method forwarded in [24].

The predictor iteration (3.3.1) usually done by a fixed-point iteration, requiring the evaluation of the integral in the last line (usually using quadrature), and then solving against the left-hand side. This fixed-point iteration in the ADER-DG Predictor turned out to be the most expensive part of the simulation. Once having chosen a basis for S_C and S_P, we introduce matrices U^n and Q^k, so that we can write

$$WQ^{k+1} = V^0 U^n - (t_{n+1} - t_n)\Psi(Q^k). \qquad (3.3.2)$$

Here, the matrices W, and V^0 are given as

$$W_{ij} = \int_T \left(w_i(t_{n+1},\vec{x}) w_j(t_{n+1},\vec{x}) - \int_{t_n}^{t_{n+1}} \partial_t w_i(t,\vec{x}) w_j(t,\vec{x})\ \mathrm{d}t \right) \mathrm{d}\vec{x}, \qquad (3.3.3)$$

$$V^0_{ij} = \int_T w_i(t_n,\vec{x}) v_j(\vec{x})\ \mathrm{d}\vec{x}. \qquad (3.3.4)$$

[7]Depending on if the area of convergence is sufficient for our equation system.

[8]Take e.g. $\boldsymbol{\mathcal{A}} \equiv 0$ to see it in (3.3.1)

The function Ψ denotes the discretized nonlinear integral from the third row in (3.3.1)

$$\Psi(Q) = (\mathrm{diag}(p)C)^T \left(\hat{\boldsymbol{\mathcal{A}}}(CQ) \cdot C^{\partial s}Q - \hat{\mathbf{S}}(CQ, C^{\partial s}Q, C^{\partial t}Q)\right). \tag{3.3.5}$$

The function $\hat{\boldsymbol{\mathcal{A}}} : \Omega^k \subseteq \mathbb{R}^{k\times 6} \to \mathbb{R}^{k\times(6\times 6)}$ means that $\boldsymbol{\mathcal{A}} : \Omega \subseteq \mathbb{R}^6 \to \mathbb{R}^{6\times 6}$ is evaluated on each row of the input, and likewise for $\hat{\mathbf{S}}$ and $\mathbf{S}$. In addition, C denotes the collocation matrix for some quadrature scheme on $[0,1]\times T$, and p are the quadrature weights. If we denote by X the matrix which has all quadrature points as rows on $[0,1]\times T$ and by X_i its i-th row, then we have for basis functions $B_P = \{w_1, ..., w_{N_P}\}$ that

$$C_{ij} = w_i(X_j). \tag{3.3.6}$$

$C^{\partial s}$ denotes the collocation tensor when applying the gradient in space. This means[9]

$$C^{\partial s} = (\partial_x w_i(X_j), \partial_y w_i(X_j)). \tag{3.3.7}$$

Likewise $C^{\partial t}$ denotes

$$C^{\partial t} = \partial_t w_i(X_j). \tag{3.3.8}$$

$\mathrm{diag}(p)$ means the matrix with p as its diagonal and zeros everywhere else.

3.3.1 Using The Tensor Product Structure

By the structure of the predictor equation (3.3.1), the tensor product structure can be carried down to the matrix level. Thus, the matrix W, can be re-written in tensor product form, since we note by writing $w_k(t,\vec{x}) = v_k(\vec{x})z_k(t)$ that

$$\begin{aligned} &W_{ij} \\ &= \int_T v_i(\vec{x})v_j(\vec{x}) \left(z_i(t_{n+1})z_j(t_{n+1}) - \int_{t_n}^{t_{n+1}} \partial_t z_i(\vec{x})z_j(t,\vec{x})\,\mathrm{d}t\right) \mathrm{d}\vec{x} \\ &= M^s_{ij} W^t_{ij}. \end{aligned} \tag{3.3.9}$$

Here M^s denotes the mass matrix in space given by

$$M^s_{ij} = \int_T v_i(\vec{x})v_j(\vec{x})\,\mathrm{d}\vec{x}, \tag{3.3.10}$$

and W^t is defined as the time-dependent part of W. Thus, we have

$$W^t_{ij} = z_i(t_{n+1})z_j(t_{n+1}) - \int_{t_n}^{t_{n+1}} \partial_t z_i(t)z_j(t)\,\mathrm{d}t. \tag{3.3.11}$$

Since the inverse commutes with the Kronecker product, we get

$$W^{-1} = (W^t)^{-1} \otimes (M^s)^{-1}. \tag{3.3.12}$$

[9]For the scope of this description, we ignore the rotation of the cells. In the code however, rotating the cells requires us to adjust the partial derivatives in space.

For the matrix V^0, we get by inserting $w_k = v_k z_k$ that

$$V^0_{ij} = z_i(t_n) \int_T v_i(\vec{x}) v_j(\vec{x}) \, \mathrm{d}\vec{x} = v^0_i M^s_{ij}, \tag{3.3.13}$$

where

$$v^0_i = z_i(t_n). \tag{3.3.14}$$

Thus, we can simplify the contribution term of U^n to

$$W^{-1} V^0 = ((W^t)^{-1} \otimes (M^s)^{-1})(v^0 \otimes M^s) = (W^t)^{-1} v^0 \otimes I. \tag{3.3.15}$$

In other words: U^n is projected in time only, but not changed in space.

We choose the quadrature rule to also have this tensor structure, and thus the collocation matrix C can be decomposed into a space and a time collocation matrix

$$C = C^t \otimes C^s. \tag{3.3.16}$$

When considering $C^{\partial s}$ and $C^{\partial t}$, we likewise obtain

$$C^{\partial s} = (C^t \otimes C^{s,\partial s}_x, C^t \otimes C^{s,\partial s}_y), \tag{3.3.17}$$
$$C^{\partial t} = C^{t,\partial t} \otimes C^s. \tag{3.3.18}$$

Note that we still use C^t and C^s here, since the derivative only gets applied to one of the components. The tensor product structure is also valid for the quadrature point weight vector for which we may write

$$p = p^T \otimes p^S. \tag{3.3.19}$$

Here, the Kronecker product is applied on vectors which again results in a vector. As a result, we may decompose

$$\operatorname{diag}(p) C = \operatorname{diag}(p^t) C^t \otimes \operatorname{diag}(p^s) C^s. \tag{3.3.20}$$

Thus, the computation of Ψ becomes:

1. Compute $(C^t \otimes C^s) Q^k$, as well as $(C^t \otimes C^{s,\partial s}) Q^k$ and $(C^{t,\partial t} \otimes C^s) Q^k$.
2. Evaluate the nonlinear function $\hat{\boldsymbol{\mathcal{A}}}$ and $\hat{\mathbf{S}}$ at each quadrature point, call the result Q'.
3. Compute $(\operatorname{diag}(p^t) C^t \otimes \operatorname{diag}(p^s) C^s) Q'$.

From the point of complexity, suppose that in space, our basis has the size n_s, and we have k^s quadrature points. In time, we have dimension n_t, and k_t quadrature points. The quadrature requires $\mathcal{O}(n_s n_t k_s k_t)$ flops without utilizing the Kronecker product. With the Kronecker product, it becomes $\mathcal{O}(n_s k_s + n_t k_t)$. Thus, if the nonlinear part of Ψ needs $f(n)$ flops for n points, we obtain $\mathcal{O}(n_s n_t k_s k_t + f(k_s k_t))$ against $\mathcal{O}(n_s k_s + n_t k_t + f(k_s k_t))$. Thus, if the nonlinear evaluations are comparably cheap, the tensor product evaluation is asymptotically more efficient as soon as the matrices are larger than (2×2).

3.3.2 Handling Bathymetry and External Source Terms

In each iteration of the predictor, we also need b evaluated at each space-time quadrature point. Since b does not change during an iteration, we can re-use the evaluations once we have computed them once.

For time-dependent bathymetry, we need to re-sample from the input in every time step. This means that given the exact bathymetry $\bar{b}$, we need to construct a projection b in our basis on $[t_n, t_{n+1}] \times T$. As we already do for the initial conditions, we interpolate and use the tensor product structure once again. In space, we use the same interpolation points as for the initial conditions. In time, we choose the Gauss-Lobatto quadrature nodes.[10] With b, we can compute ∇b and $\partial_t b$ over the collocation matrices.

3.4 The Corrector Step in Strong Form

Given a solution $\mathbf{q}$ for the predictor iteration (3.3.1), as well as the previous estimate $\mathbf{u}^n$ and the next step $\mathbf{u}^{n+1}$, we have for each test function $\mathbf{v} \in S_C$ that

$$\begin{aligned}
&\int_T \mathbf{v}(\vec{x})^T \mathbf{u}^{n+1}(\vec{x})\,\mathrm{d}\vec{x} \\
=& \int_T \mathbf{v}(\vec{x})^T \mathbf{u}^{n}(\vec{x})\,\mathrm{d}\vec{x} \\
&- \int_{[t_n,t_{n+1}]\times\partial T} \mathbf{v}(\xi)^T (\mathcal{N}(\mathbf{q}, \mathbf{q}^+(\xi)) \cdot n(\xi))\,\mathrm{d}(t, S(\xi)) \\
&+ \int_{[t_n,t_{n+1}]\times T} \mathcal{F}(\mathbf{q}) \cdot \nabla \mathbf{v}(\vec{x})\,\mathrm{d}(t,\vec{x}) \\
&- \int_{[t_n,t_{n+1}]\times T} \mathbf{v}(\vec{x})^T \left(\boldsymbol{\mathcal{B}}(\mathbf{q}) \cdot \nabla \mathbf{q} + \mathbf{S}(\mathbf{q}, \nabla \mathbf{q})\right)\mathrm{d}(t,\vec{x}).
\end{aligned} \tag{3.4.1}$$

Here, the third row contains a surface integral. $n(\xi)$ denotes the outward normal, and $\mathbf{q}^+(\xi)$ the value from the adjacent cell at a boundary point ξ.

In comparison to the predictor iteration, we only use test functions in space. After integrating by parts in time just as for (3.3.1), we again replace $\mathbf{q}(t_n, x)$ by $\mathbf{u}^n$. But additionally, we also insert $\mathbf{q}(t_{n+1}, x)$ for $\mathbf{u}^{n+1}$, and we applied Gauss' Theorem to introduce the boundary flux terms (cf. the third and fourth row in (3.4.1)). Note that this only modifies the conservative part of the PDE—but it also means that we take the influence of neighboring cells into account. The detailed derivation is shown in [29] or e.g. [7]. It shall be noted that as before, the usage of the test functions differs again to [29] and [7], but we can again show that the formulations are equivalent under certain conditions.

When evaluating the corrector equation as described in (3.4.1), we once again need to evaluate $\mathbf{q}$ at quadrature points, then evaluate all the integrals, and sum the points up

[10] We shall note that it *may* be possible to avoid interpolating $\bar{b}$ at least in time: the expression $\partial_t b$ in (3.3.1) can be removed using integration by parts. Then, we only need to evaluate $\bar{b}$ at t_n and t_{n+1}, and at all other quadrature points in time. We have not tried nor implemented this idea.

again. This procedure is almost identical to a single predictor iteration step—however, we have already found a fixed point at this time, thus effectively we do no additional work. But, since the corrector and the predictor equation are structurally similar, we can simply re-cast the solution $\mathbf{q}$ to the predictor iteration to obtain $\mathbf{u}^{n+1}$ by a mere linear transformation.

To begin, we transform (3.4.1) to the so-called "strong form" [22]. Using Gauss' Theorem again yields

$$\begin{aligned}\int_T \mathbf{v}(\vec{x})^T \mathbf{u}^{n+1}(\vec{x})\,\mathrm{d}\vec{x} &= \int_T \mathbf{v}(\vec{x})^T \mathbf{u}^{n}(\vec{x})\,\mathrm{d}\vec{x} \\ - \int_{[t_n,t_{n+1}]\times\partial T} &\mathbf{v}(\xi)^T (\mathcal{N}(\mathbf{q},\mathbf{q}^+(\xi)) - \mathcal{F}(\mathbf{q}))\cdot n(\xi))\,\mathrm{d}(t,S(\xi)) \\ - \int_{[t_n,t_{n+1}]\times T} &\mathbf{v}(\vec{x})^T \left(\boldsymbol{\mathcal{A}}(\mathbf{q})\cdot\nabla\mathbf{q} + S(\mathbf{q},\nabla\mathbf{q})\right)\mathrm{d}(t,\vec{x}).\end{aligned} \tag{3.4.2}$$

This time, we do not replace the boundary integral over $\mathbf{F}$ by the numerical flux. As a result, we can now insert the numerical fluctuations $\mathcal{D} = \mathcal{N} - \mathcal{F}$ in the second line.

The strong form in (3.4.2) still requires us to evaluate a volume integral, but this can be removed under the condition of a weak assumption. Under a very slight abuse of notation, we can write the following.

Assumption 3.1. *We have $S_C \subseteq S_P$, i.e. for $\mathbf{v} \in S_C$, there is a $\mathbf{w} \in S_P$, such that $\mathbf{w}(t,\vec{x}) = \mathbf{v}(\vec{x})$ for all t and $\vec{x}$.*

This means that the predictor function space contains each possible space function frozen in time. If we assume a tensor product structure for S_P, then this assumption means that the time function space S_T contains the constant functions.[11]

We recall now that $\mathbf{q}(t,\vec{x})$ fulfills the predictor equation (3.3.1) for all $\mathbf{w} \in S_P$. Given Assumption 3.1, it therefore holds in particular for all $\mathbf{w} = \mathbf{v} \in S_C$. In this case, we obtain for the predictor

$$\begin{aligned}\int_T \mathbf{v}(\vec{x})^T \mathbf{q}(t_{n+1},\vec{x})\,\mathrm{d}\vec{x} &= \int_T \mathbf{v}(\vec{x})^T \mathbf{u}^{n}(\vec{x})\,\mathrm{d}\vec{x} \\ - \int_{[t_n,t_{n+1}]\times T} &\mathbf{v}(\vec{x})^T \left(\boldsymbol{\mathcal{A}}(\mathbf{q})\cdot\nabla\mathbf{q} - S(\mathbf{q},\nabla\mathbf{q})\right)\mathrm{d}(t,\vec{x}),\end{aligned} \tag{3.4.3}$$

since the time-dependent terms vanish. Inserting this equation into the strong corrector equation (3.4.2), we obtain

$$\begin{aligned}\int_T \mathbf{v}(\vec{x})^T \mathbf{u}^{n+1}(\vec{x})\,\mathrm{d}\vec{x} &= \int_T \mathbf{v}(\vec{x})^T \mathbf{q}(t_{n+1},\vec{x})\,\mathrm{d}\vec{x} \\ - \int_{[t_n,t_{n+1}]\times\partial T} &\mathbf{v}(\xi)^T \left(\mathcal{D}(\mathbf{q},\mathbf{q}^+(\xi))\cdot n(\xi)\right)\mathrm{d}(t,S(\xi)).\end{aligned} \tag{3.4.4}$$

[11]As it is the case when choosing e.g. the monomial basis in time.

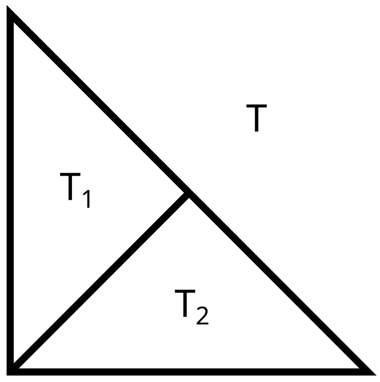


Figure 3.1: The scenario for the adaptive mesh refinement. We are given an orthogonal triangle T which is split into two orthogonal triangles T_1 and T_2.

As a result, we only need to evaluate the predictor solution $\mathbf{q}$ at time t_{n+1} against the space basis.[12] In particular, we are able to avoid a nonlinear volume integral evaluation by this step.

When converting to matrices, we can once again make use of the tensor product structure.

$$M^s U^{n+1} = (V^1)^T Q - (\Delta t)\Psi_F(Q). \tag{3.4.5}$$

Here, the matrix V^1 is defined as

$$V^1_{ij} = \int_T w_i(t_{n+1}, \vec{x}) v_j(\vec{x}) \, \mathrm{d}\vec{x} = v^1_i M^s_{ij}. \tag{3.4.6}$$

The vector v^1 is given similarly to v^0 as

$$v^1_i = z_i(t_{n+1}). \tag{3.4.7}$$

Then we get again that the mass matrix cancels out due to the tensor product structure, and we are left with

$$(M^s)^{-1}(V^1)^T = (v^1)^T \otimes I. \tag{3.4.8}$$

The integration term over the numerical fluctuations need the same treatment as the collocation matrices in the predictor.

3.5 Adaptive Mesh Refinement

We also added support for adaptive mesh refinement. This process was already started in [29], but it was ultimately not finished.

[12]Given that the construction of the corrector involves replacing $\mathbf{q}(t_{n+1}, x)$ with $\mathbf{u}^{n+1}(x)$, this result does in hindsight almost seem to be expected—if Assumption 3.1 is fulfilled.

In the situation given in sam(oa)^2, we consider a triangle T split in two halves T_1 and T_2, as shown in Figure Figure 3.1. To project our basis from T onto T_1 and T_2, we use an L^2 projection, i.e. we compute for $k = 1, 2$ the matrix

$$\frac{1}{2} M^s A^k = B^k, \tag{3.5.1}$$

where the matrix B^k is defined as

$$B^k_{ij} = \int_{T_k} \psi^k_i \varphi_j \, \mathrm{d}\vec{x}. \tag{3.5.2}$$

Here, the (φ_i) are the basis functions from B_C transformed to T, and (ψ^k_i) denote the basis functions transformed to T_k. The mass matrix M^s of T is given by

$$M^s_{ij} = \int_T \varphi_i \varphi_j \, \mathrm{d}\vec{x}, \tag{3.5.3}$$

and we have

$$\int_{T_k} \psi^k_i \psi^k_j \, \mathrm{d}\vec{x} = \frac{|T_k|}{|T|} \int_T \varphi_i \varphi_j \, \mathrm{d}\vec{x} = \frac{1}{2} M^s_{ij}. \tag{3.5.4}$$

Here $|T|$ denotes the volume of T, and $|T_k|$ the volume of T_k. We utilize that $|T| = |T_1| + |T_2|$, and $|T_1| = |T_2|$.

For the projection from T_1 and T_2 back to T, we compute the Moore-Penrose inverse of

$$A = \begin{pmatrix} A^1 \\ A^2 \end{pmatrix}. \tag{3.5.5}$$

Thus, denoting the Moore-Penrose inverse by $A^\dagger$, we obtain

$$A^\dagger A = I. \tag{3.5.6}$$

This means: if we project from T to T_1 and T_2, and then back to T immediately, we get the same coefficients as before.

Furthermore, we need to choose a criterion by which we decide to refine or to coarsen. Inspired by [15], we look at a formulation which uses the idea of the variance from probability theory, as well as the concept of total variation. For that, we look at the finite volume representation. Given a set of edges $\{e_1, ..., e_k\}$ and adjacent cells $c_{i,1}, c_{i,2}$, we compute the discrete total variation

$$v_1 = \frac{1}{k} \sum_{i=1}^{k} |c_{i,1} - c_{i,2}|, \tag{3.5.7}$$

and weigh it by the number of edges.[13] We also compute the squared total variation

$$v_2 = \frac{1}{k} \sum_{i=1}^{k} (c_{i,1} - c_{i,2})^2. \tag{3.5.8}$$

[13]We note that not all edges in our finite volume representation have the same length; but we have chosen to neglect this for now.

By Jensen's inequality,[14] we have that

$$v_1^2 \leq v_2. \tag{3.5.9}$$

Next, we choose two parameters $a_{\text{ref}} \leq a_{\text{cor}}$, so that for

$$v_1^2 \leq a_{\text{cor}} v_2, \tag{3.5.10}$$

we coarsen the cell. For

$$v_1^2 \geq a_{\text{ref}} v_2, \tag{3.5.11}$$

we refine the cell. In all other cases, we keep it as it is.

3.6 An Updated Finite Volume Scheme

Lastly, we updated the finite volume scheme to accommodate the changes described in the previous sections. In [29], we used a Godunov splitting[15] scheme for the hyperbolic part and the source term. In particular, we had for a value U^n that

$$U^{n+1} = (\mathcal{S}_{\Delta t} \circ \mathcal{V}_{\Delta t})(U^n). \tag{3.6.1}$$

The finite volume operator is given as

$$\mathcal{V}_{\Delta t}(U) = U - \frac{\Delta t}{\Delta x} \int_{\partial T} \mathcal{N}(U, U^+(\xi)) \cdot n(\xi) \, \mathrm{d}S(\xi). \tag{3.6.2}$$

In essence, this is the ADER-DG method when using constant functions in time and space. Here Δx is the length of the shortest edge of T. The source operator $\mathcal{S}_{\Delta t}(U)$ is given as the solution V of the implicit Euler method

$$V = U + (\Delta t)\mathbf{S}(V, 0). \tag{3.6.3}$$

The exact formula is given in [29].

Numerical Fluctuations Switching from numerical fluxes to numerical fluctuations did not require us to make any changes in the finite volume scheme: this is due to assuming a constant value U for each patch, and the integral of a constant over a closed boundary being zero. Thus, we have for any triangle T that

$$\int_{\partial T} \mathcal{F}(U) \cdot n(\xi) \, \mathrm{d}S(\xi) = 0. \tag{3.6.4}$$

[14]As a reminder: Jensen's inequality can be used to prove that $\operatorname{Var}(X) \geq 0$ for a square-integrable random variable X. Using $\operatorname{Var}(X) = \mathbb{E}\left(X^2\right) - \mathbb{E}(X)^2$, or rather, $\mathbb{E}\left(X^2\right) \geq \mathbb{E}(X)^2$, gave us the idea.

[15]Alternatively called Lie-Trotter splitting

As a result, we obtain that

$$\begin{aligned}
&\int_{\partial T} \mathcal{D}(U, U^+(\xi)) \cdot n(\xi) \, \mathrm{d}S(\xi) \\
&= \int_{\partial T} \big(\mathcal{N}(U, U^+(\xi)) - \mathcal{F}(U)\big) \cdot n(\xi) \, \mathrm{d}S(\xi) \\
&= \int_{\partial T} \mathcal{N}(U, U^+(\xi)) \cdot n(\xi) \, \mathrm{d}S(\xi) - \int_{\partial T} \mathcal{F}(U) \cdot n(\xi) \, \mathrm{d}S(\xi) \\
&= \int_{\partial T} \mathcal{N}(U, U^+(\xi)) \cdot n(\xi) \, \mathrm{d}S(\xi).
\end{aligned} \tag{3.6.5}$$

Therefore, we may simply replace $\mathcal{N}$ by $\mathcal{D}$ in the formulation of $\mathcal{V}_{\Delta t}$ to get

$$\mathcal{V}_{\Delta t}(U) = U - \frac{\Delta t}{\Delta x} \int_{\partial T} \mathcal{D}(U, U^+(\xi)) \cdot n(\xi) \, \mathrm{d}S(\xi). \tag{3.6.6}$$

Time-Dependent Bathymetry We extend the splitting scheme to handle the time-dependent bathymetry. Integrating

$$\partial_t(hp) = 2c^2 \partial_t b \tag{3.6.7}$$

over $[t_n, t_{n+1}]$ gives

$$(hp)(t_{n+1}) = (hp)(t_n) + 2c^2(b(t_{n+1}) - b(t_n)). \tag{3.6.8}$$

We combine this with the previous steps from our numerical scheme as described in [29], and thus we obtain the new scheme as

$$U^{n+1} = (\mathcal{P}_{\Delta t} \circ \mathcal{S}_{\Delta t} \circ \mathcal{V}_{\Delta t})(U^n). \tag{3.6.9}$$

In short, we now apply three operators which are applied one after another: first the finite volume scheme, then the source term $\mathbf{S}$ without the pressure correction, and finally the pressure correction (3.6.8).

4 Changes to sam(oa)^2

We continued working on the implementation from [29], and development continued on the `dswe` branch of the sam(oa)^2 repository[30].

4.1 sam(oa)^2

sam(oa)^2[19] stands for "Space-Filling Curves and Adaptive Meshes for Oceanic And Other Applications", and it is written in FORTRAN 90. The program implements a structured grid over a space-filling curve (SFC). The main operations in sam(oa)^2 are grid traversals which operate independently on each cell, edge, and node of the grid. For that, sam(oa)^2 takes care of the distribution and the parallelization of the traversal using MPI and OpenMP. Thus, each implemented application only needs to take care of their respective logic for the cells, edges, and nodes and optimize them for a single thread.

The space-filling curve in sam(oa)^2 yields a quadratic domain, and orthogonal triangular cells. By refining the space-filling curve, a triangle can be split into two as shown in Figure 3.1. Depending on how often we do this for the whole mesh, we can speak of different subdivision levels N_D. We denote the number of triangles by N_M. At $N_D = 0$, we have only $N_M = 2$ triangles which form a square. At $N_D = 1$, we have $N_M = 4$ triangles, and for a general level N_D, we have $N_M = 2^{N_D+1}$ triangles.

So far, different linear equation system solvers[19], or equation systems like the heat equation, Darcy's equation or a finite volume implementation of the shallow water equations[10] have been implemented in sam(oa)^2. An implementation of the ADER-DG method for the shallow water equations (2.2.1) was forwarded in [25]. In [29], we implemented the system (2.2.2) using the ADER-DG method. The code currently resides on the `dswe` branch of the sam(oa)^2 repository[30]. For each timestep, the grid is traversed twice: the first traversal computes the ADER-DG predictor on each cell and transfers the resulting data onto the edges of the cells. The second traversal calculates the numerical fluxes with the projected data and completes the corrector step of the ADER-DG method. This traversal also computes the next timestep. The implementation also includes a finite volume limiter (called on demand in the second traversal) and the Discrete Maximum Principle. However, there was only minimal support for adaptive mesh refinement and no support for time-dependent bathymetry until this work.

4.2 Changes to the Implementation

The aderdg app of sam(oa)^2 was adapted to include the enhancements described in the previous chapters. This includes the improvements which we made to the ADER-DG method, the updates to the finite volume scheme, as well as support for AMR.

4.2.1 Scenario Implementation and ASAGI

In comparison to the previous implementation, the scenario[1] interface was reworked. All functions are assumed to be marked with `elemental` or `impure elemental`, i.e. they operate on scalars, but the operations can be broadcasted to arrays.

ASAGI [26] is used for parsing and evaluating earthquake data. It had already been included in the `FVM` app of sam(oa)^2, where it replaced the built-in scenario initial conditions. In the current ADERDG-DSWE simulation, ASAGI is included as a scenario itself instead. Thus, we were able to remove the dependency to ASAGI from the core ADERDG code. The functions of the ASAGI scenario are marked as `impure elemental` which allows them to call ASAGI element-wise. The command line arguments are the same as for the `FVM` app in sam(oa)^2.

4.2.2 Including YaTeTo

To speed up the matrix computations, we replaced the FORTRAN `matmul` calls with YaTeTo [33] kernels. YaTeTo is Matrix kernels are specified in Python using the Einstein Summation Notation. That is, for a matrix-vector multiplication $c = Ab$ of a matrix $A \in \mathbb{R}^{m\times n}$, and a vector $b \in \mathbb{R}^n$, we have the computation

$$c_i = \sum_{j=1}^{n} A_{ij} b_j. \tag{4.2.1}$$

The Einstein Summation Notation allows us to omit the summation sign, since we implicitly assume a sum over all indices which are not on the right-hand side. Thus, we obtain

$$c_i = A_{ij} b_j. \tag{4.2.2}$$

In YaTeTo, this is written as

```
c['i'] <= A['ij'] * b['j']
```

This assumes that we have previously defined `A`, `b`, and `c` as tensors. The `<=` symbol is overloaded from Python to indicate an assignment to a tensor. Three- or higher-dimensional tensors can be indexed likewise.

In order to deal with scalar variables, we re-interpret them as zero-order tensors. Thus, they are written as `a['']` for a scalar a.

[1]In scons, this was the parameter `swe_scenario` for the `SWE` application.

The Kronecker product structure can be translated directly: given matrices $A \in \mathbb{R}^{n_1 \times m_1}$ and $B \in \mathbb{R}^{n_2 \times m_2}$, and $C \in \mathbb{R}^{m_1 \times m_2}$; write $c = \text{vec}(C)$, i.e. c is C written as a vector. Then $d = (A \otimes B)c$ is written in the Einstein Summation Notation, i.e.

$$D_{i_1 i_2} = A_{i_1 j_1} B_{i_2 j_2} C_{j_1 j_2}. \tag{4.2.3}$$

In YaTeTo, this becomes

```
D['iI'] <= A['ij'] * B['IJ'] * C['jJ']
```

A broadcast operation, i.e. the introduction of a new tensor dimension, could be accommodated by a multiplication with the vector $i = (1, ..., 1)^T$ which contains only the number 1. If we have a matrix `A`, the vector i named `ones`, and an output tensor of dimension 3 called `B`, then broadcasting `A` to `B` can be done by

```
B['ijk'] <= A['ij'] * ones['k']
```

Broadcasting is needed for converting a space-only vector to a predictor (i.e. space-time) vector.

So far, YaTeTo only has an interface for C++ on its current master branch. While there is a FORTRAN interface, it does not seem to be up-to-date with the latest developments. Hence, we wrote our own adapter for YaTeTo to FORTRAN which mimics the behavior of the existing interface. It exports all functions into a FORTRAN module which is subsequently included into all other files.

In addition, we needed to define a file which exports the symbols `libxsmm_num_total_flops` and `pspamm_num_total_flops`, as YaTeTo did not define them in the generated code, but referenced them nonetheless.

4.2.3 Adaptive Mesh Refinement

The mesh refinement was implemented using the adaptive traversal template in sam(oa)^2. That is, the space-filling curve is changed according to given flags. For each cell, we have one of the following cases, depending on a given refinement flag.

- A cell stays the same. In this case, we can simply copy the data.
- A cell is refined. An adaptive traversal pass allows for a refinement to at most two levels, i.e. one cell can become four cells at most.
- A cell is coarsened. This can only be done for one level, i.e. one cell can only be combined with the cell which lies next to it in the space-filling curve.[2]

Thus, we can handle at most one mesh subdivision per traversal. Moreover, the adaption traversal assures that the mesh stays conform, i.e. an edge is shared between only two triangles at most at all times.

[2]The refinement flag of this next cell is subsequently ignored.

```
Input: A SFC $S_I$ containing all cells
Input: $r_v$: the number of cells to be refined
Input: $r(c)$: indicates refinement for each cell $c$. Either $r(c) = 1$ for
       refinement, $r(c) = -1$ for coarsening, or $r(c) = 0$.
Output: An output SFC $S_O$
$S \leftarrow S_I$;
while $r_v > 0$ do
    Start an empty SFC $S'$;
    $r_v \leftarrow 0$;
    forall cells $c$ in $S$ do
        if $r(c) \leftarrow 1$ then
            Create child cells $c_1, ..., c_n$ and add them to $S'$;
            forall $i = 1, ..., n$ do
                Project $c$ onto $c_i$;
                if (3.5.11) is true for $c_i$ then
                    $r(c_i) \leftarrow 1$;
                    $r_v \leftarrow r_v + 1$;
                end
                $r(c_i) \leftarrow 0$;
            end
        else if $r(c) = -1$ then
            Take the next cell $c_D$ from $S$;
            Merge the two cells to a cell $c_C$ and add it to $S'$;
            Project $c$ and $c_D$ to $c_C$;
            if (3.5.10) is true for $c_C$ then
                $r(c_C) \leftarrow -1$;
                $r_v \leftarrow r_v + 1$;
            end
            $r(c_C) \leftarrow 0$;
        else
            Copy $c$ to $S'$;
        end
    end
    $S \leftarrow S'$;
end
$S_O \leftarrow S$;
```

Algorithm 4.1: The algorithm for multiple iterations of the adaptive mesh refinement procedure. Each iteration of the outer while loop executes one traversal.

Whether a coarsening or refinement is needed is specified in a traversal before the adaption. That is in our case, the corrector traversal: at the point where the next timestep is computed, we also check the refinement criterion. After the corrector, the adaption traversal is called. The basic logic for a single adaption traversal is similar to the `FVM` [10] implementation.

Especially in the context of time-dependent bathymetry, it is sometimes useful to refine multiple levels at the same time. This is for example the case, if at some point during the simulation, we have a more complex or small ocean floor movement appearing. Therefore, after refining or coarsening a cell, we check the adaption criterion again. If we want to refine a cell again, we re-run the adaption traversal. This process is repeated until there are no more cell refinements requested in either direction. However, we need to consider the possibility of cycling: suppose that in one step we refine a cell, but in the next step, the same cell is marked for coarsening again. We end up at the same grid with what we started, and by the construction of the projection operators, we also have the same cell values. As a result, we refine again, only to coarsen in the next pass. To solve this problem, we only allow a cell to be refined, if it has been previously refined (but not if it was coarsened). Likewise, we only allow the further coarsening, if a cell has previously been coarsened.

Moreover, the time stepping needs to be re-evaluated once the adaption traversal was completed, this is due to the cell size being changed.

The complete procedure is skipped entirely, if the corrector traversal requests no refinements. Then, we proceed with the predictor step immediately instead.

See Algorithm 4.1 for an overview over the complete procedure after calling the corrector traversal.

4.2.4 Convergence Measuring Traversal

We implemented an additional traversal to output convergence-related data. For this, we assume that the scenario is known over the whole time frame, so that we can evaluate the scenario functions for all values of t and $\vec{x}$.

We can then write for a (scalar) function u and its reference solution $\bar{u}$ that

$$\|u - \bar{u}\|_p = \left(\int_\Omega (u(\vec{x}) - \bar{u}(\vec{x}))^p \,\mathrm{d}\vec{x}\right)^{\frac{1}{p}} = \left(\sum_{T\in\mathcal{T}} \int_T (u(\vec{x}) - \bar{u}(\vec{x}))^p \,\mathrm{d}\vec{x}\right)^{\frac{1}{p}} \tag{4.2.4}$$

for $p \in [1, \infty)$. Thus, we can compute the error by computing the integral

$$\int_T (u(\vec{x}) - \bar{u}(\vec{x}))^p \,\mathrm{d}\vec{x} \tag{4.2.5}$$

on each triangle T. This can be done by simply traversing all cells, and the integrals can be computed using a high-order quadrature rule. For a vector-valued function, this can be done for each of its components.

The resulting values are formatted into a JSON file, and the convergence traversal is called by request on each output step.

4.3 Changes to the Repository Structure

Apart from the implementation, we made some general changes to the sam(oa)^2 repository and the build environment. These were done to mainly ease the development. Once again, they only reside on the `dswe` branch for now.

4.3.1 Code Reorganization

The code of sam(oa)^2 was reorganized into a new folder structure, to make the code more easy to navigate. For that, the sam(oa)^2 scenarios (i.e. equations) were re-named to sam(oa)^2 apps, to avoid confusion with the SWE or ADER-DG scenarios (i.e. initial data) which were also called “scenarios”.

The code is still contained in `src`, but it now contains the following sub folders.

- `include` contains all files which are included via preprocessor directive. This includes all generic data structures, as well as the traversals.
- `core` contains the sam(oa)^2 files which are needed regardless of the application run on top of it. This includes mostly the grid and space-filling curve logic.
- `lib` contains code to connect to other libraries, or parts of code which are not needed by all apps.
- `app` contains the code of each of the apps, as well as the code to run them.

Most of the previous folders only contained a single app (e.g. cppcodeSWE or `Darcy`), and were therefore moved to the `app` folder.

Temporarily, two files are still shared between all apps; these are `Config.f90` which the command line argument parsing logic, as well as `SFC_traversal.f90` which contains the main methods for all apps.

4.3.2 Build System

The new code uses CMake [14] as build system, as compared to scons [11] before. However, GNU Makefiles *cannot* be used for building sam(oa)^2, as it fails to parse the non-preprocessed FORTRAN files for dependencies correctly. As a replacement, ninja [21] can be employed.

Currently, the limitation that only one app can be built at the same time is still in place; since the grid data types do not support templates so far.

With scons, all code files were compiled regardless if they contained code for the currently activated app or not, and activated over macros when necessary. This was changed to only compiling the code files which belong to the active app. For that, each app has a CMake file in its main folder located in `src/app`.

For detecting libraries such as `libxsmm` [13], we used the CMake files from the SeisSol repository [31]. Thus, we also introduced the same notation for the host architectures from SeisSol.

4.3.3 Compiler Support

The compilers `flang` and `nvfortran`[3] are supported now as well. More specifically, we tested the compilers AOCC flang 3.2.0, and NVHPC nvfortran 22.09.

This required us to fix a bug in the `stream` data structure which would cause segmentation faults otherwise. The initialization and subroutine overloading of some types had to be fixed as well. For `nvfortran`, the cross-compilation with C++ caused a linker error, since the FORTRAN main function is not recognized by the `nvc++` compiler which CMake applied for linking. This problem could be fixed by setting the linker executable for C++ to `nvfortran` explicitly.

[3]Formerly known as `pgfortran`

5 Evaluation

In this chapter, we present results computed with sam(oa)^2 and the H-BMSS-γ system (2.3.8). The list of scenario names which are present in sam(oa)^2 is given in Table 5.1.

Since the evaluation was mostly focused on testing and verifying the model, we computed the results on a laptop with a Ryzen 4800H processor (8 cores, 2 threads per core), and 32 GiB RAM. As operating system, we used Linux in version `6.0.12-arch1-1` (Arch Linux distribution). As a compiler, we used GCC (including `g++` and `gfortran`) in version 12.2.0. Furthermore, we used the `rome` hardware configuration, and the `libxsmm_jit` backend for YaTeTo.

5.1 Model Tests

We begin with some simple test scenarios. These are meant for verifying our implementation, as well as testing the behavior of time-dependent bathymetry for the system (2.3.8).

5.1.1 Convergence Analysis

To begin with, we re-do the convergence test which was already done in [29]. This time, we use a manufactured solution analysis instead, using the solitary wave for the system (2.2.2) for $c \to \infty$, as described in Chapter 2. That is, we look at equation (2.2.19). We

Name	Description
`QUASI_SOLITARY_WAVE`	Eq. (2.2.19), expanded in the y direction
`FLAT1`	Eq. (5.1.9), $k = 1$
`FLAT2`	Eq. (5.1.9), $k = 2$
`FLAT3`	Eq. (5.1.9), $k = 3$
`CYLINDER`	Eq. (5.2.1)
`GAUSS1`	Eq. (5.3.2) with $A = 1$, $\sigma_r = 12.5\,\mathrm{km}$, $\sigma_t = 125\,\mathrm{s}$
`GAUSS2`	Eq. (5.3.2) with $A = 1$, $\sigma_r = 12.5\,\mathrm{km}$, $\sigma_t = 1.25\,\mathrm{s}$
`GAUSS3`	Eq. (5.3.2) with $A = 1$, $\sigma_r = 1.25\,\mathrm{km}$, $\sigma_t = 1.25\,\mathrm{s}$
`ASAGI`	Bathymetry taken from ASAGI [26]

Table 5.1: The scenarios with their corresponding name in sam(oa)^2.

write

$$\bar{\mathbf{q}}(t,x,y) = \begin{pmatrix} h(\xi(t,x)) \\ h(\xi(t,x)) \cdot u(\xi(t,x)) \\ 0 \\ h(\xi(t,x)) \cdot w(\xi(t,x)) \\ h(\xi(t,x)) \cdot p(\xi(t,x)) \end{pmatrix}. \tag{5.1.1}$$

The functions h, u, w, p are taken from (2.2.19). ξ is defined as in (2.2.16). In essence, $\bar{\mathbf{q}}$ is the representation in conserved variables, and extended in the y direction. Using $\bar{\mathbf{q}}(t,x,y)$, we compute the function $\mathbf{z}(t,x,y)$ defined as

$$\mathbf{z} = \partial_t \bar{\mathbf{q}} + \boldsymbol{\mathcal{A}}(\bar{\mathbf{q}}) \cdot \nabla \bar{\mathbf{q}} - \mathbf{S}(\bar{\mathbf{q}}, \nabla \bar{\mathbf{q}}). \tag{5.1.2}$$

We computed $\mathbf{z}$ using the sagemath [27] computer algebra system by inserting (2.2.19), and converted it to FORTRAN using sympy [20]. In [29], we compared the solitary wave solution for $c \to \infty$ (i.e. system (2.2.19)) to the quasi-exact solution of the solitary wave for finite values of c. As expected, we obtain system (2.2.19) for $c \to \infty$. Thus, if we choose c to be large enough, the influence of $\mathbf{z}$ should be minimal. Using $\mathbf{z}$, we solve the PDE

$$\partial_t \mathbf{q} + \boldsymbol{\mathcal{A}}(\mathbf{q}) \cdot \nabla \mathbf{q} = \mathbf{S}(\mathbf{q}, \nabla \mathbf{q}) - \mathbf{z} \tag{5.1.3}$$

in sam(oa)^2 for $\mathbf{q}$. By the construction of $\mathbf{z}$, we have the solution $\mathbf{q} = \bar{\mathbf{q}}$,[1] but we cannot represent $\bar{\mathbf{q}}$ without error in our function basis on the given grid. Therefore, we will get an error during the simulation.

We ran our code for polynomials of degree 1 to 6. This is more than in [29], where we only tested up to degree 4. For each degree, we test the subdivisions 4 to 13. As parameters for (2.2.19), we chose $A = 0.005$ to have a smooth solution, so that the interpolation error of the initial conditions is manageable. The basic water height is set to $H = 1$. Furthermore, $\gamma = 2$ and $c = 3\sqrt{gH} \approx 9.40$. For the boundary condition, we enforce the exact solution. We used a domain of 50×50, and positioned the peak of the wave at $x = 12.5$ of it. The time is chosen so that the peak of the wave is positioned at $x = 37.5$ at the end of the simulation. In short, we let the wave travel over half of the domain. Given our value for A, this was $T_{\text{end}} \approx 7.962\,\text{s}$. We wrote an output file at 26 timestamps, i.e. $\Delta t = T_{\text{end}}/25 \approx 0.3185\,\text{s}$.

Two-Dimensional Convergence Analysis

We first use the built-in convergence traversal described in Section 4.2.4.

Figure 5.1 shows the results for h at the last timestep. There seemed to be an accuracy barrier below 10^{-11}, so we marked it with a line. For all other variables, the picture looked similar; therefore, we do not offer extra plots for them.

In Table 5.2, we give the convergence orders. Most of the time, we have for polynomial degree N_P only the order N_P. For $N_P = 6$, we even get an order of 7 in the area above the accuracy barrier. Thus, we reach a high-order convergence for our implementation.

[1]We silently assume here that the solution to (5.1.3) is unique, but we do not prove it.

N_P	h	hu	hv	hw	hp
1	1.43	1.41	1.45	1.36	1.20
2	2.00	2.02	1.81	2.01	1.81
3	2.63	2.74	2.83	2.52	2.38
4	3.72	3.86	3.62	3.94	3.83
*5	4.98	5.20	5.44	4.84	4.79
*6	7.45	7.50	6.79	7.43	7.19

Table 5.2: The empirical convergence order w.r.t. L^2 norm, measured in two dimensions, as shown in Figure 5.1. The values were determined by a linear regression over the logarithmic values. All orders marked with a star were only computed on all values where the error for h was larger than 10^{-11} to rule out errors due to numerical barriers.

The convergence order is approximately the same in all variables. This should not seem to be surprising; only hv is of special interest. Ideally, we should have $hv \equiv 0$; however, the triangular cells cause small imbalances at their corners and edges, as it can be seen in Figure 5.3 for one example.

In Figure 5.2, we show the error for each of the 26 output timestamps. The timestamp 0 therefore effectively shows the initial interpolation error. Thus, the error seems to be mostly dominated by the interpolation of the initial conditions. The first point to notice is that the mesh subdivisions always lie pairwise next to each other, i.e. an even and the next-higher uneven subdivision lie closer to each other than an uneven subdivision and the next-higher even subdivision. Furthermore, we can observe a slight wave-resembling pattern in the error, in the lower subdivisions like 4 or 5. Finally, we note the behavior for the subdivision 13 for the degrees $4, 5, 6$, as well as 12 for $5, 6$ and 10 and 11 for 6, i.e. that the L^2 error slowly rises. Our conjecture is that this is related to the accuracy barrier we observe. Furthermore, for degrees 5 and 6, the said subdivision show a stronger increase after timestamp 0, effectively nullifying the advantage of the higher subdivisions in the case of degree 6.

We should note that we also tried to do the whole two-dimensional convergence test with $A = 0.2$ as in [7]. However, here the interpolation of the initial condition on our grid caused large inaccuracies for the subdivisions up to 13.

One-Dimensional Convergence Analysis

Secondly, we re-visit the one-dimensional analysis from [29] and examine it more closely. Here, we take a line through the quadratic domain and reconstruct the polynomials along this line. After that, we compare against the one-dimensional reference solution given by (2.2.19). We should be able to see convergence in this case as well, we already run into an accuracy barrier at around $4 \cdot 10^{-6}$. In Figure 5.4, we show the convergence for the degrees 1 to 3; other than the one-dimensional setting, the situation is the same as for the two-dimensional convergence test. It should be noted that for higher degrees,

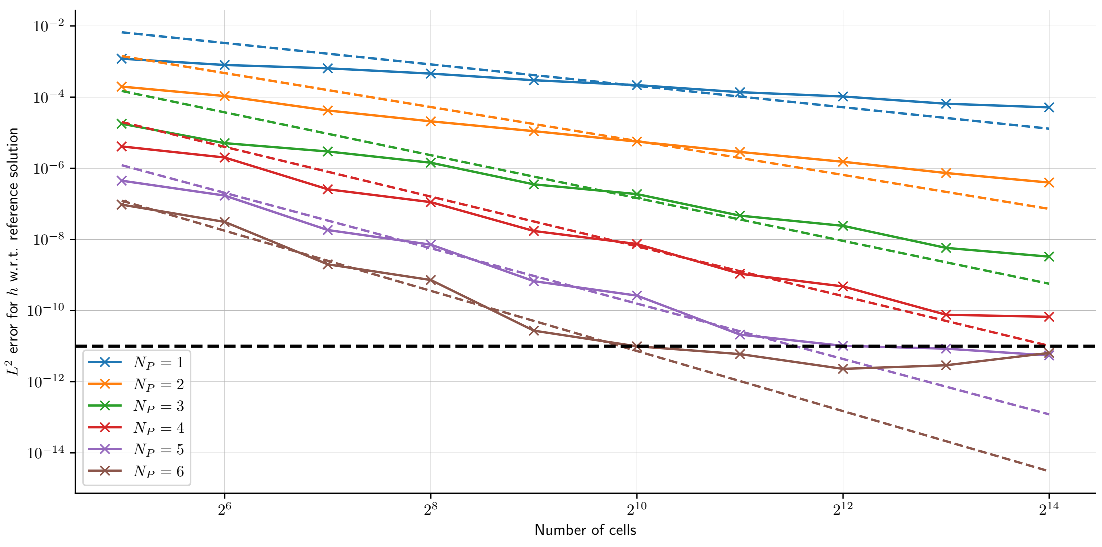


Figure 5.1: The empirical convergence order w.r.t. L^2 norm, measured in two dimensions by a convergence traversal. The values were determined by a simulation of (2.2.19) with (2.2.2) using a manufactured solution, and compared against the reference solution. The black dashed line at 10^{-11} shows the accuracy barrier, below which we deem values too inaccurate to be considered for the calculation of the empirical convergence. The crosses show the actually measured values; the solid lines are for visual clarity only. The dashed lines show the empirical convergence order for each degree, as determined by a linear regression over each value higher than the accuracy barrier at 10^{-11}. For subdivision N_D, we have $N_M = 2^{N_D+1}$.

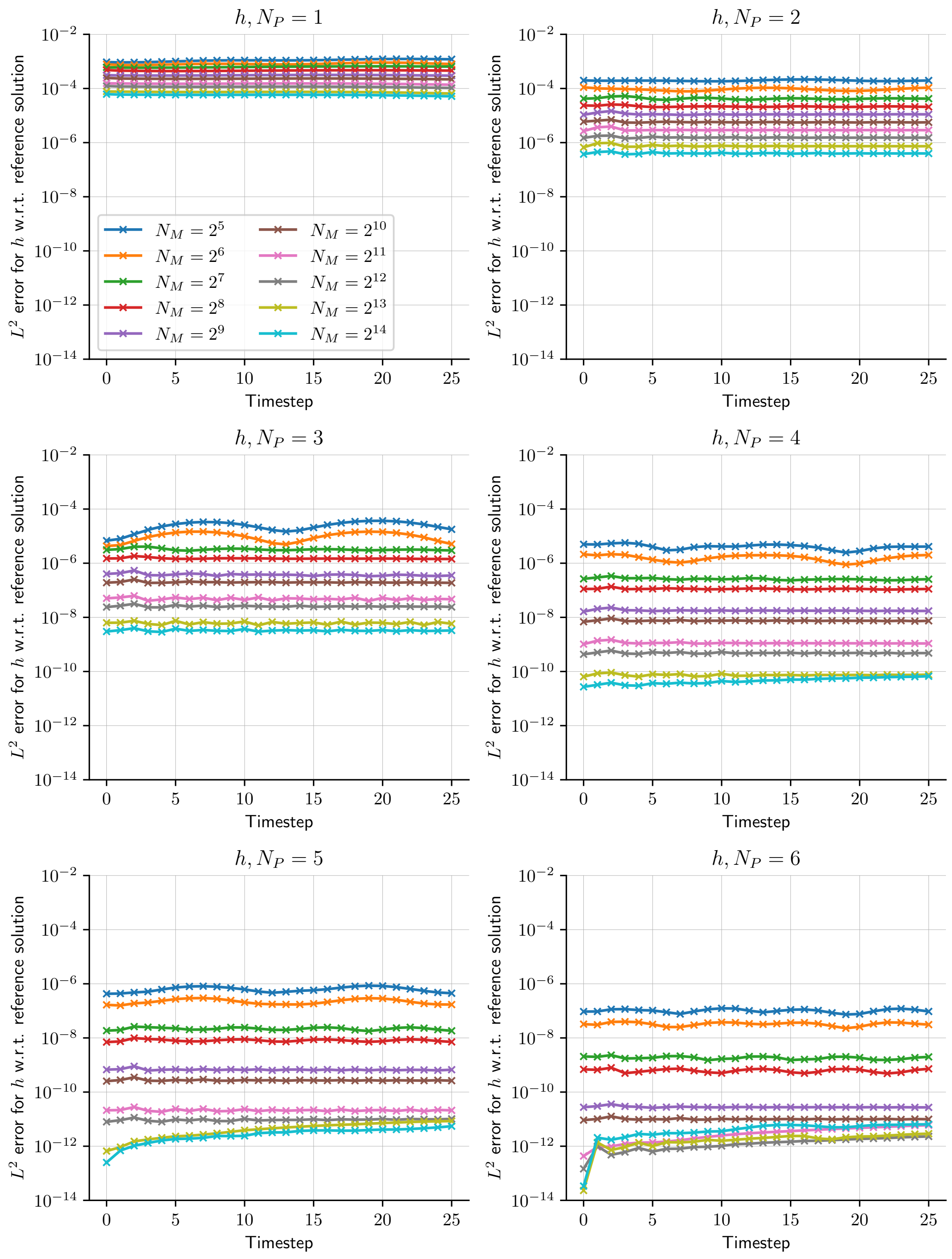


Figure 5.2: The error w.r.t. L^2 norm, measured in two dimensions by a convergence traversal over time for (2.2.19) against the manufactured solution. The timestep is $\Delta t \approx 0.3185\,\mathrm{s}$.

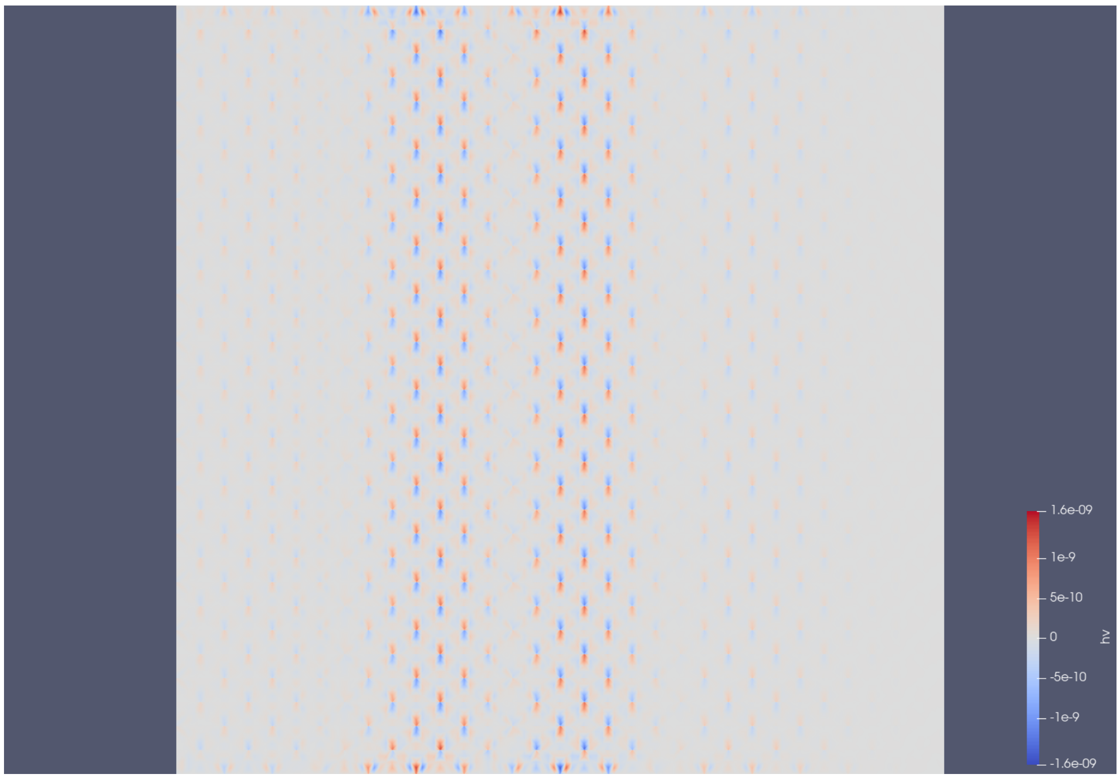


Figure 5.3: The values of hv (ideally, it should be zero 0). We chose polynomials of degree $N_P = 4$, a mesh subdivision $N_D = 10$, and the time 3.185 s.

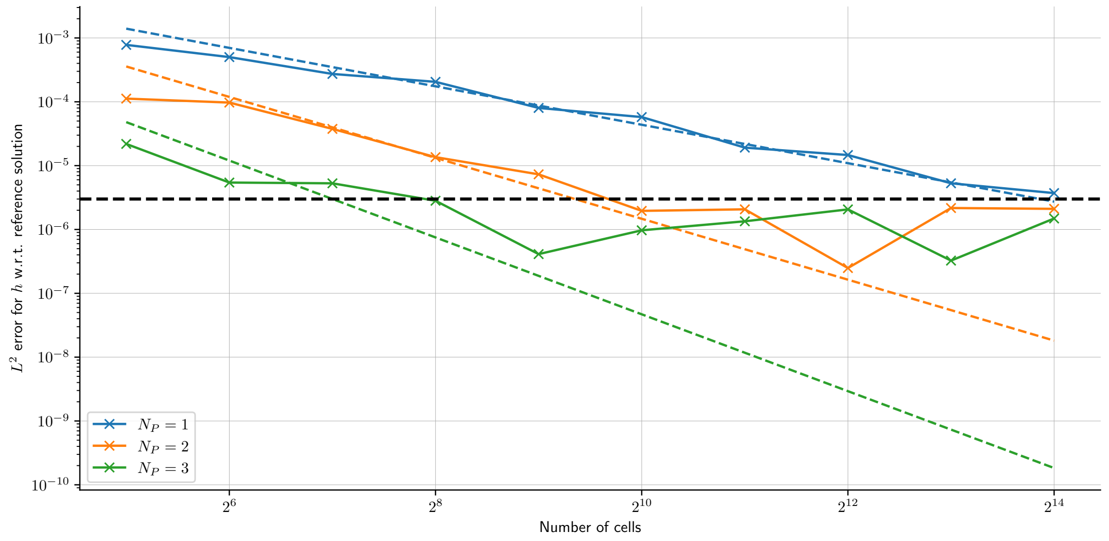


Figure 5.4: The empirical convergence order w.r.t. L^2 norm, measured in one dimension. The values were determined by a simulation of (2.2.19) with (2.2.2) using a manufactured solution, and compared against the reference solution. The black dashed line at $4 \cdot 10^{-6}$ shows the accuracy barrier, below which we deem values too inaccurate to be considered for the calculation of the empirical convergence. The crosses show the actually measured values; the solid lines are for visual clarity only. The dashed lines show the empirical convergence order for each degree, as determined by a linear regression over each value higher than the accuracy barrier at $4 \cdot 10^{-6}$. For subdivision N_D, we have $N_M = 2^{N_D+1}$.

we stay below the accuracy barrier in almost all cases, and hence we omitted them in the plot. Figure 5.5 shows the error over time. Already for the higher mesh subdivisions for degree 2 polynomials, we observe that we reach the accuracy barrier—which only occurred around 10^{-11} for the two-dimensional analysis. When looking at the difference between the computed and the exact solution, we see a smooth error function which means that the computed wave is slightly tilted in the moving direction. This error function shown persists throughout the simulation, and a similar function can be seen for all other orders. Its amplitude slightly changes or oscillates over time, but in general, it does not increase noticeably. A similar type of problem with a similar error function was already noticed (but not shown) in [29] for the quasi-exact solution used there.

5.1.2 Bathymetry Changes

In order to test the effect of moving bathymetry in the most simple cases, we use the equations derived in Section 2.3.2. That is, we take a still ocean and a flat bathymetry,

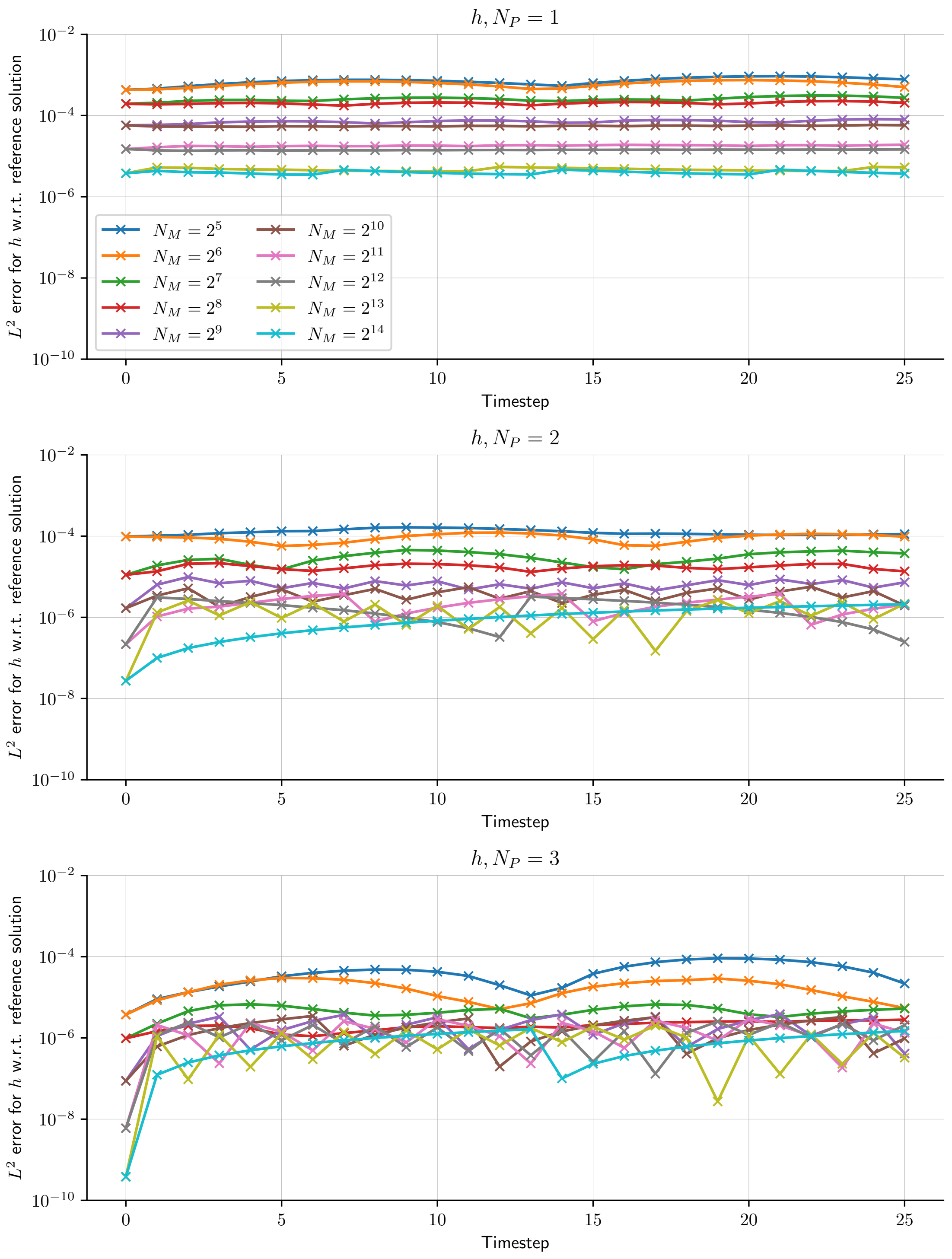


Figure 5.5: The error w.r.t. L^2 norm, measured in one dimension by a convergence traversal over time for (2.2.19) against the manufactured solution. The timestep is $\Delta t \approx 0.3185\,\mathrm{s}$.

N_P	h
1	1.85
*2	2.10
*3	2.04

Table 5.3: The empirical convergence order w.r.t. L^2 norm, measured in one dimension, as shown in Figure 5.4. The values were determined by a linear regression over the logarithmic values. All orders marked with a star were only computed on all values where the error for h was larger than $4 \cdot 10^{-6}$ to rule out errors due to numerical barriers.

and move it upwards.[2] In short, we consider the following conditions:

$$h(0,\ \cdot\ ) \equiv 1, \tag{5.1.4}$$
$$u(0,\ \cdot\ ) \equiv 0, \tag{5.1.5}$$
$$v(0,\ \cdot\ ) \equiv 0, \tag{5.1.6}$$
$$w(0,\ \cdot\ ) \equiv 0, \tag{5.1.7}$$
$$p(0,\ \cdot\ ) \equiv 0. \tag{5.1.8}$$

For b, we prescribe the following function. For a given k, we define

$$b(t) = \max\left\{t^k, 1\right\}. \tag{5.1.9}$$

To compute the solution for this b analytically, we take (2.3.24), (2.3.27) for the time frame $[0, 1)$, since b is differentiable in this region. For $t < 1$ and $k = 1$, we have already computed the solution in Section 2.3.2 and obtained (2.3.30) and (2.3.29). Hiegher k can be reached by using integration by parts, until all polynomial terms vanish. For $t \geq 1$, we take the previously-computed solution, and use its continuous extension to $t = 1$.[3] This gives us the values $p(1)$ and $h(1)$ which we subsequently use as initial conditions for the ODE (2.3.14) with $\partial_t b \equiv 0$, i.e. we have the homogenous case. Alternatively, we may use integration by parts on (2.3.24) and (2.3.27) to replace $\partial_t b$ by b.

In Figure 5.6, we show the results for $k = 1, 2, 3$. As it can be seen, the results conform with the analytic structure: for $t < 1$, we observe that both w and p show an oscillatory, but for $k = 2, 3$ also an increasing motion. In fact, p shows a similar pattern for $k + 1$ as w for k. In the case $k = 1$, the phase and the amplitude are exactly as we computed with (2.3.30) and (2.3.29) when inserting $h = 1$, $C = 1$, and $c = 4$. For $t \geq 1$, we observe an un-dampened oscillation, as predicted in the previous paragraph. We also note that w is differentiable at $t = 1$, while p is only continuous, but not differentiable there.

[2] Effectively, we could speak of a point (i.e. zero-dimensional) domain.

[3] This is possible, because with the given b, the solutions (2.3.24), (2.3.27) produce a continuous and bounded function on $[0, 1)$.

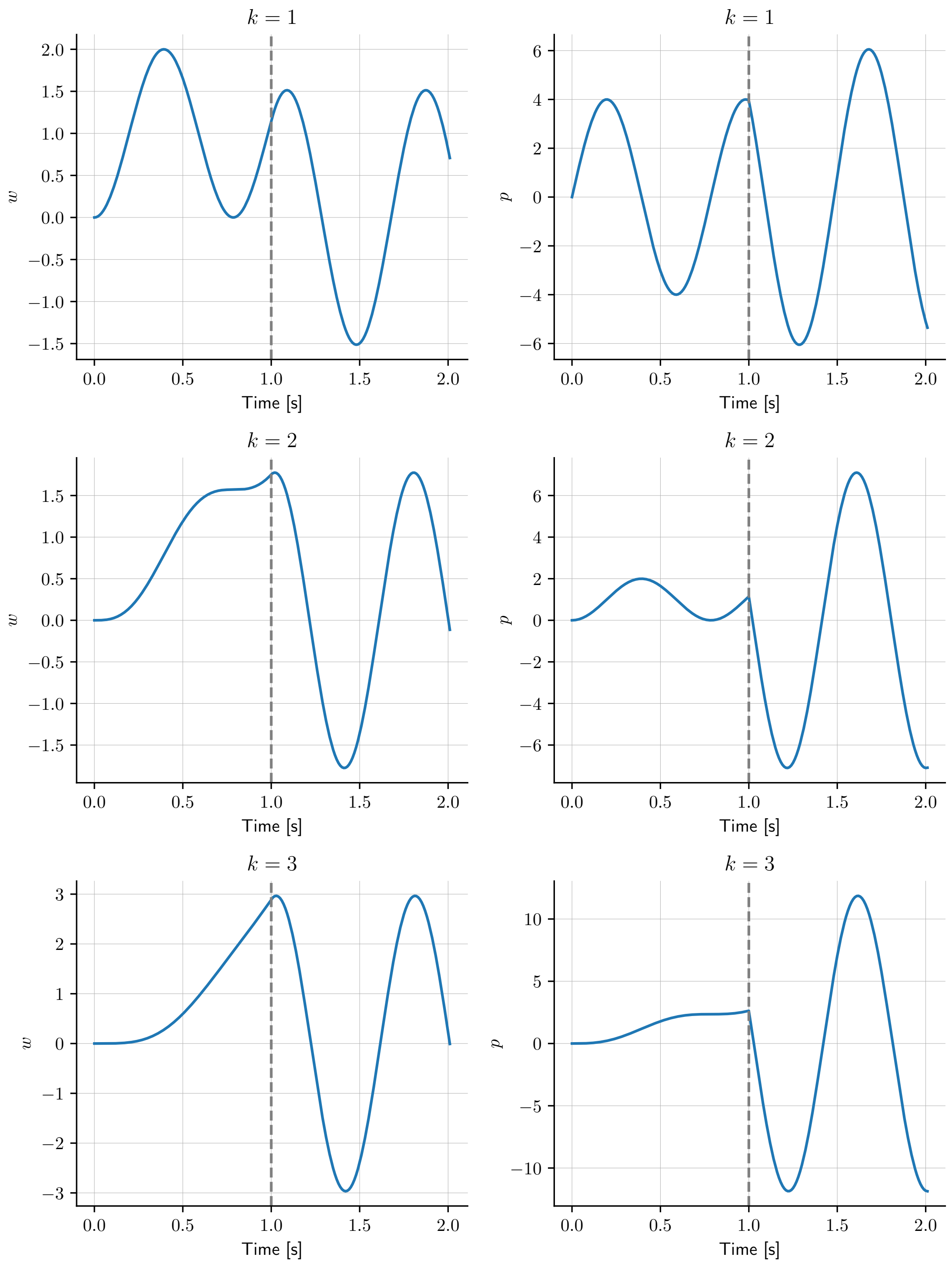


Figure 5.6: The bathymetry (5.1.9) for $k = 1, 2, 3$ on a zero-dimensional domain, i.e. h, u, and v are constant, and $h \equiv 1$. We have $\gamma = 2$, as well as $c = 4$. We show the values for $w = hw$ and $p = hp$. The subdivision of the domain is set to 1, and the polynomial degree to $N_P = 1$.

5.2 Adaptive Mesh Refinement

For showing the AMR performance of our code, we choose the following setup: given a domain of 10 km × 10 km, we consider the time-dependent bathymetry (units in meter)

$$b(t, x, y) = \begin{cases} \max\{2t, 1\} & \sqrt{x^2 + y^2} < 2000 \\ 0 & \text{otherwise} \end{cases} \tag{5.2.1}$$

The water height is set to 1 km; i.e. we start with a flat and still ocean. We simulate for 30 s. The parameters are set to $c = 100$ and $\gamma = 2$. The mesh subdivision is allowed to range from 6 to 12. The polynomial degree is $N_P = 4$. We additionally set the AMR parameters to $a_{\text{cor}} = 0.8$ and $a_{\text{ref}} = 0.5$. If $v_2 < 10^{-6}$, we also coarsen.

Since the uplift of the ocean floor happens very quickly, we observe that the water in the simulation behaves as in a dam-break scenario, i.e. a Riemann Problem. The structure of these for (2.3.8) was discussed in [29]. In short, we observe three different waves: a fast outwards-travelling shock wave, a quasi-stationary contact discontinuity which separates two areas with different non-hydrostatic pressure, as well as a rarefaction wave in the center. In particular, the rarefaction, since we are in a two-dimensional setting, is expected to collapse very quickly once it reaches the center.

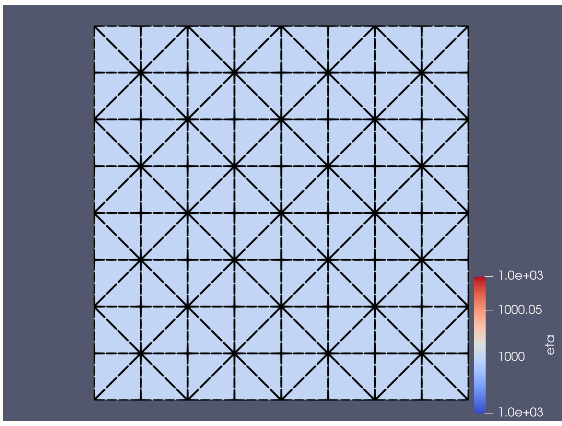
In Figure 5.7, we see the first few milliseconds of the simulation. As soon as the cylinder causes uplift, we see that the AMR becomes active around the new edges in the water level. Moreover, we have already multiple cell refinements once we reach $t = 10$ ms. Near $t = 50$ ms, the shock wave begins to separate from the rest of the cylinder. However, the center of the cylinder stays at a slightly coarser level, since the rarefaction has not reached it yet.

The seconds after the first expansion, are shown in Figure 5.8. We see more cells getting refined, as the shockwave travels through the domain. Also, the rarefaction wave causes more refinement in the center and begins to collapse. However, as soon as it has travelled far enough, some cells between the shockwave and the contact discontinuity get coarsened.

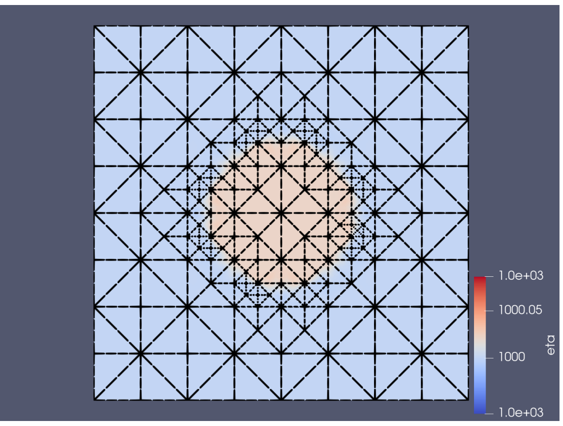
This trend becomes more apparent in the next seconds, as seen in Figure 5.9. The shockwave has by now reached the border of the domain and gets reflected, but in between, we see that cells are being coarsened. In the center, the rarefaction has fully collapsed, oscillates, and then begins to send out a second wave. This causes the center of the cylinder to become flat again, so we observe coarsening effects here as well. Until this second wave arrives, the contact discontinuity still causes the cells to be refined to a high subdivision.

5.3 Tsunami Simulation

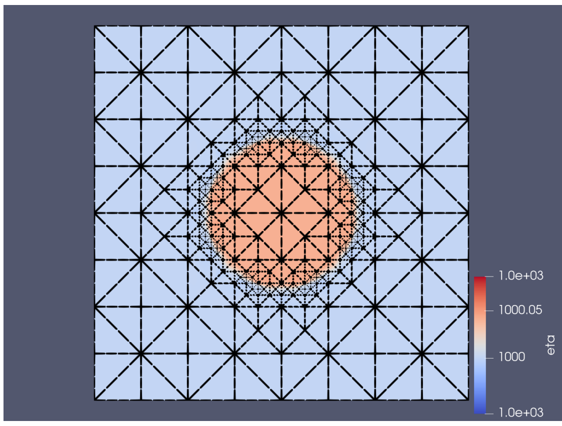
Lastly, we evaluate the models which we introduced in Chapter 2 together with a change in space, unlike the model scenario from Section 2.3.2.



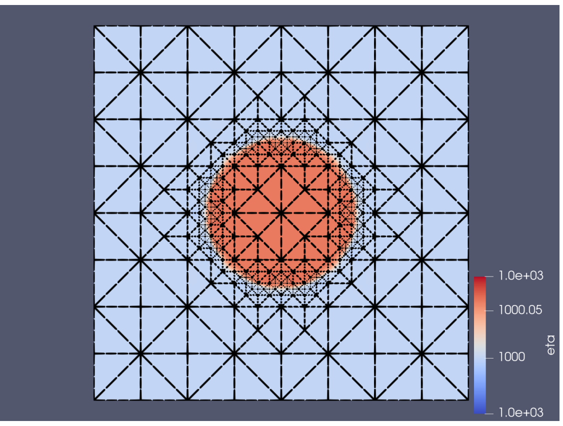
(a) $t = 0\,\mathrm{ms}$



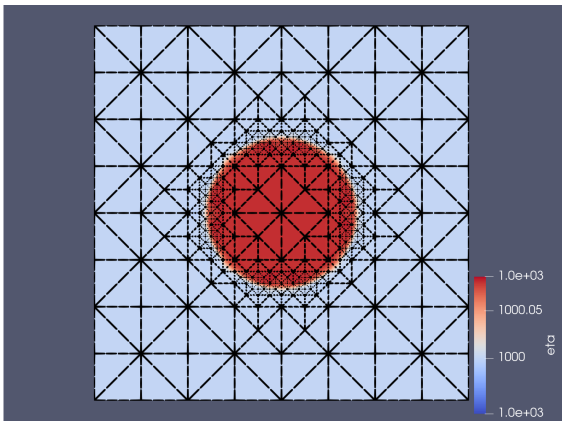
(b) $t = 10\,\mathrm{ms}$



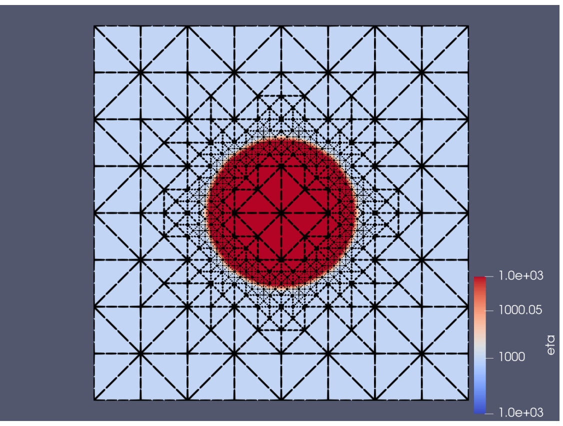
(c) $t = 20\,\mathrm{ms}$



(d) $t = 30\,\mathrm{ms}$

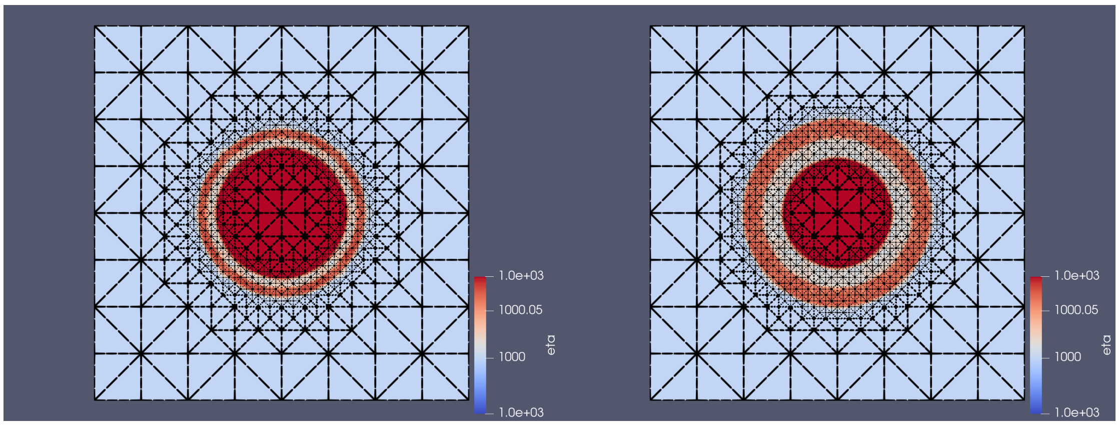


(e) $t = 40\,\mathrm{ms}$

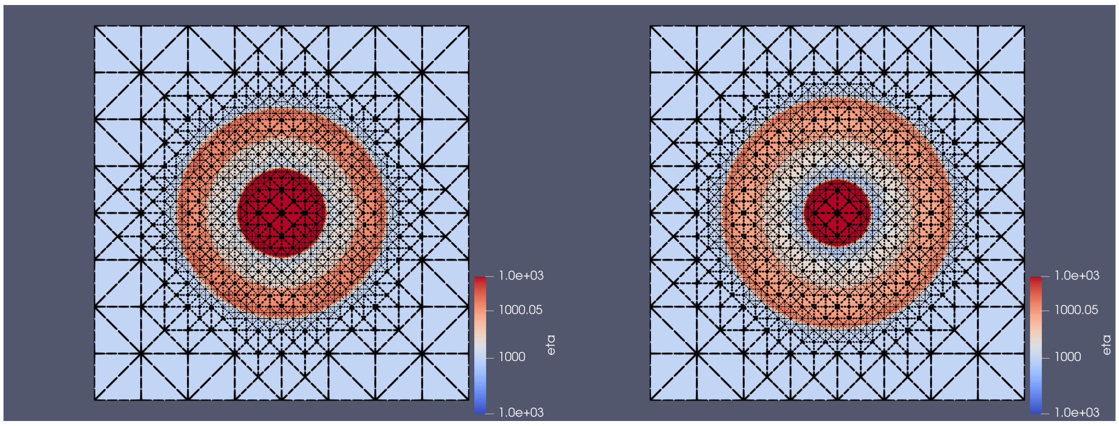


(f) $t = 50\,\mathrm{ms}$

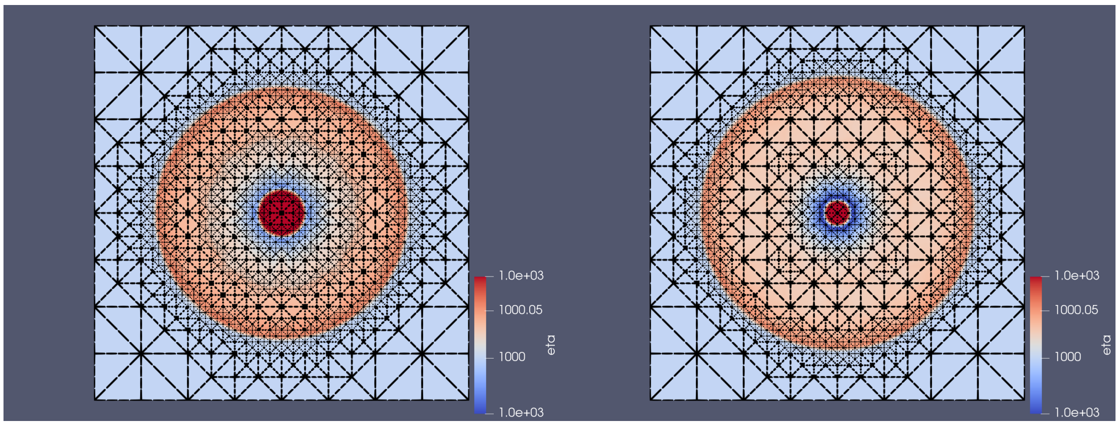
Figure 5.7: The simulation of a cylindrical uplift with adaptive mesh refinement. The black lines indicate the current cells. Simulation time 0 ms to 50 ms. The simulation was done with polynomials of degree 4 in space and time, $c = 100$ and $\gamma = 2$. The water height is set to 1 km. The bathymetry (5.2.1) consists of a small rising cylinder.



(a) $t = 2\,\mathrm{s}$ (b) $t = 4\,\mathrm{s}$

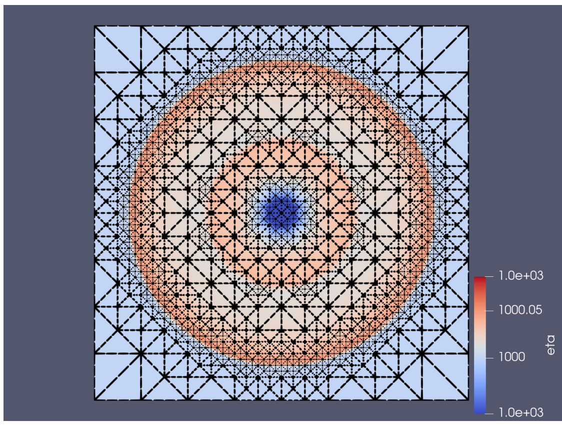


(c) $t = 6\,\mathrm{s}$ (d) $t = 8\,\mathrm{s}$

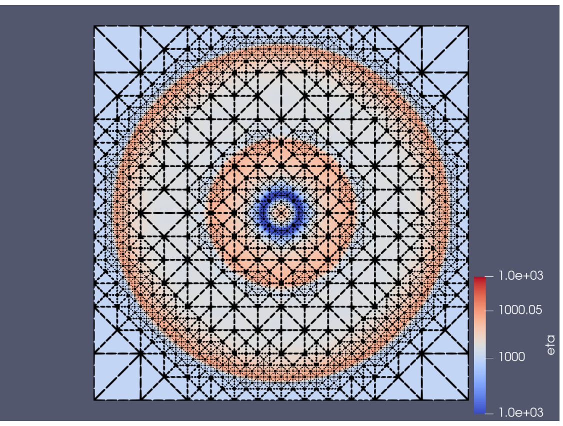


(e) $t = 10\,\mathrm{s}$ (f) $t = 12\,\mathrm{s}$

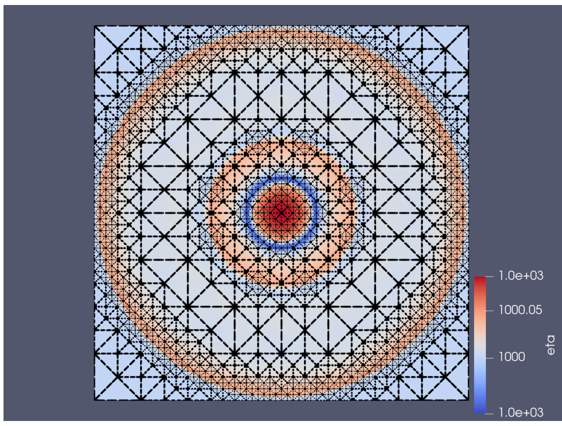
Figure 5.8: The simulation of a cylindrical uplift with adaptive mesh refinement. The black lines indicate the current cells. Simulation time 2 s to 12 s. The simulation was done with polynomials of degree 4 in space and time, $c = 100$ and $\gamma = 2$. The water height is set to 1 km. The bathymetry (5.2.1) consists of a small rising cylinder.



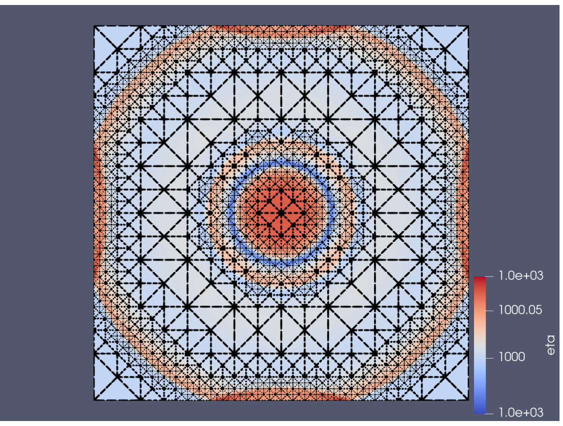
(a) $t = 15\,\mathrm{s}$



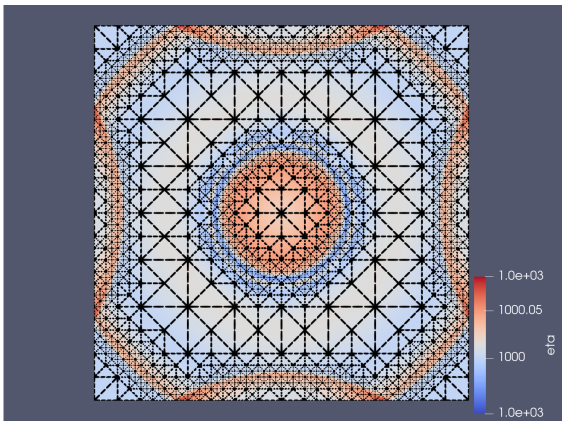
(b) $t = 18\,\mathrm{s}$



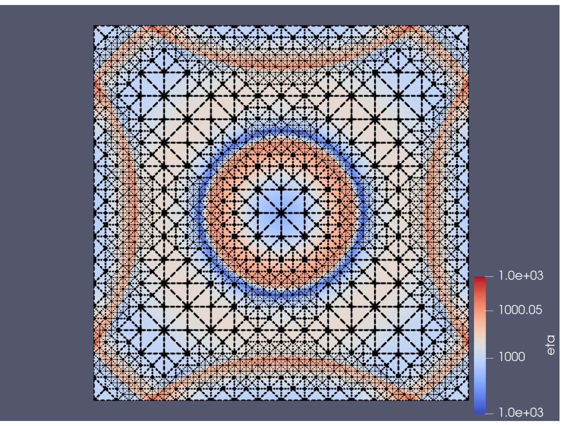
(c) $t = 21\,\mathrm{s}$



(d) $t = 24\,\mathrm{s}$



(e) $t = 27\,\mathrm{s}$



(f) $t = 30\,\mathrm{s}$

Figure 5.9: The simulation of a cylindrical uplift with adaptive mesh refinement. The black lines indicate the current cells. Simulation time 15 s to 30 s. The simulation was done with polynomials of degree 4 in space and time, $c = 100$ and $\gamma = 2$. The water height is set to 1 km. The bathymetry (5.2.1) consists of a small rising cylinder.

5.3.1 Gaussian Uplift

We begin by comparing our three methods from Table 2.1 in the three test scenarios which are presented in [1]. In each of these, we have

$$\partial_t b(t,x,y) = \frac{A}{\sigma_t\sqrt{2\pi}} \exp\left(-\left(\frac{(t-4\sigma_t)^2}{2\sigma_t^2}\right) - \left(\frac{x^2+y^2}{2\sigma_r^2}\right)\right) \tag{5.3.1}$$

Here σ_t and σ_r are model parameters which control the speed and shape of the earthquake, respectively. The parameter A controls the amplitude of the earthquake. Since our current implementation in sam(oa)^2 only supports the initialization on b, and not $\partial_t b$, we need to integrate in time. Assuming that $b(0,x,y) = 0$ for all (x,y), we obtain

$$b(t,x,y) = A \cdot \left(\Phi\left(\frac{t-4\sigma_t}{\sqrt{2}\sigma_t}\right) - \Phi(-2\sqrt{2})\right) \cdot \exp\left(-\left(\frac{x^2+y^2}{2\sigma_r^2}\right)\right), \tag{5.3.2}$$

where Φ is the distribution function of the Standard Normal Distribution given as

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{s}{2}\right) \mathrm{d}s. \tag{5.3.3}$$

In our code, we use the `erf` function for Φ. The parameters for the three test cases are given in Table 5.1 by the scenarios `GAUSS1`, `GAUSS2`, and `GAUSS3`.[4] They can also be found in [1]. In short, scenario 1 (`GAUSS1`) only shows the main tsunami wave, scenario 2 (`GAUSS2`) adds acoustic waves, and scenario 3 (`GAUSS3`) adds dispersion to the main tsunami wave and keeps the acoustic waves.

We simulate all three test cases using methods 1a, 1b, and 2. As parameters for (2.3.8), we chose $c = 1000$, and $\gamma = 2$. The choice of $c = 1000$ is motivated by the fact that with a characteristic water height of $h_0 \approx 4000\,\mathrm{km}$ and $g \approx 10\,\mathrm{m\,s^{-2}}$, we want $c = \alpha\sqrt{gh_0} \approx 200\alpha$ and $\alpha \geq 1$. The idea behind this choice is to let the error wave travel faster than the main tsunami wave (cf. [7]). According to [7], choosing $\alpha = 5$ is supposed to provide a good approximation to the original model for $c \to \infty$.

The results for the water level displacement is shown in Figure 5.10. For scenario 1, the results of method 1a and 2 conform with the results shown for the shallow water equations in [1], when compared to the respective initialization. Method 1b shows slightly distorted results. While the scenario has negligible pressure effects, the instantaneous pressure build-up causes slight oscillations which faintly resemble acoustic effects, but they have (to our knowledge) no physical meaning. We are going to call them error-correcting waves. For scenario 2, we see that method 1a shows the same results as in scenario 1 which is to be expected, since the initial conditions are exactly the same, since we have the same value for σ_r. Method 2 also shows the same behavior as the shallow water method, however small pressure oscillations are visible. While these are not as strong as shown in [1], they roughly have the same frequency. Furthermore, method 1b this time roughly matches the results of method 2.

[4]The preprint of [1] which we worked with did not include a value for A. However, $A = 1$ yielded the same results as shown in [1] when comparing the resulting plots.

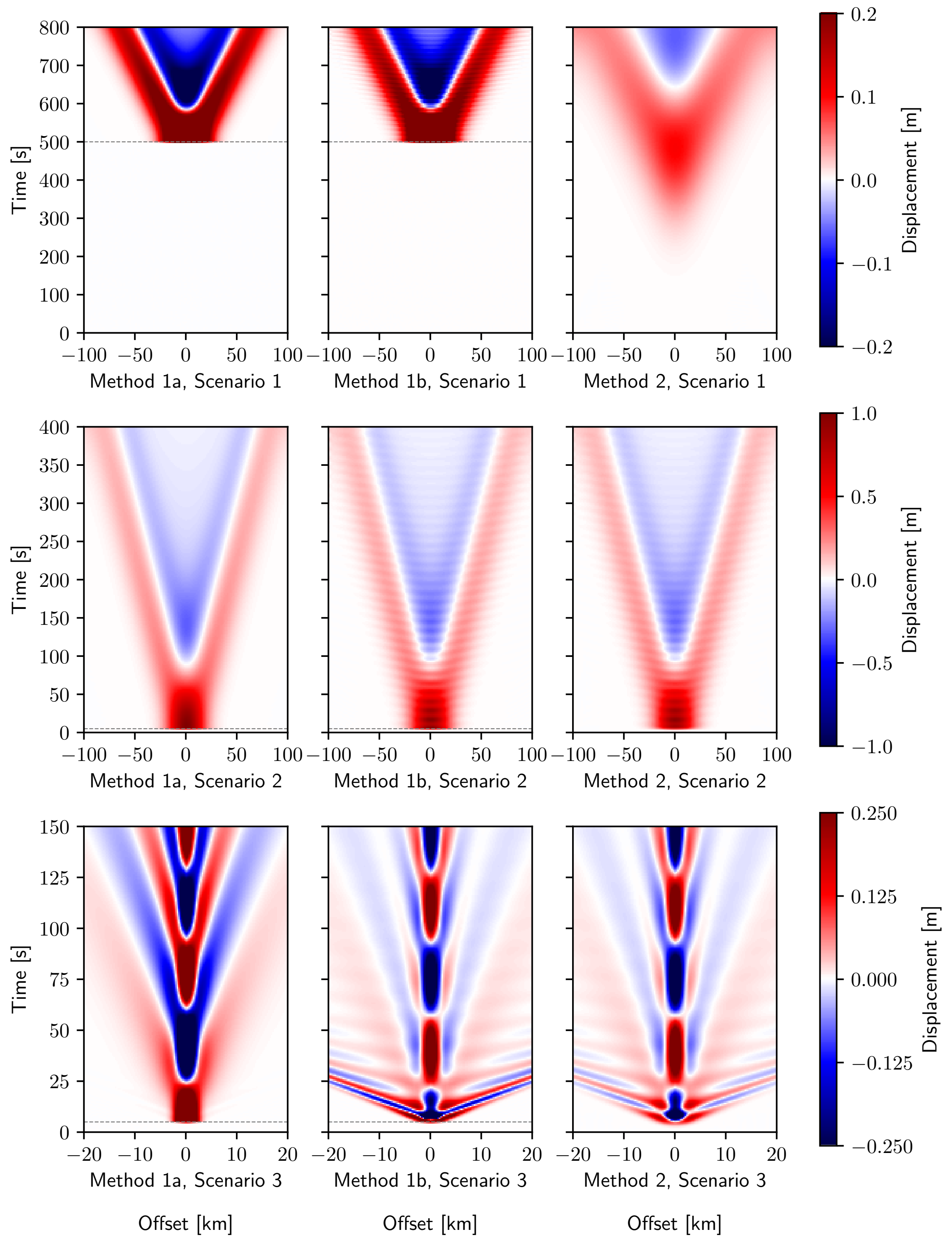


Figure 5.10: Scenarios 1,2, and 3 from [1] simulated with methods 1a, 1b, and 2 from Table 2.1 and (2.3.8). The polynomial degree was set to $N_P = 4$, and the subdivision to $N_D = 8$ for scenarios 1 and 2, and to $N_D = 12$ for scenario 3. The constants were set to $c = 1000$ and $\gamma = 2$.

In scenario 3, we see the biggest changes compared to the original methods. The center oscillates strongly for all three methods. These oscillations seem to be the source of the surrounding dispersion which is visible from around 5 km from the center onwards. Moreover, the tsunami wave propagates seemingly slower than expected for the shallow water case (cf. Figure 5.13, the image for $c = 0$), and it also seems to dissipate. While this may look like an inaccuracy at first, a similar effect is visible in [1] for scenario 3 and the fully-coupled model (method 1, i.e. [1, Figure 5(i)]). However, our methods seems to indicate that the phase wave accelerates over time—whereas the fully-coupled model shows a visually constant phase velocity nonetheless. This may be related to the central oscillations pushing water mass and pressure waves outwards. When comparing the three methods among each other, we notice that method 1b and method 2 yield fairly similar results. The only noticeable difference is that method 1b overshoots the displacement in the beginning which conforms with the effects seen in the other scenarios for method 1a and 1b. In contrast to method 2 and method 1b, method 1a yields a different result. Firstly, the oscillation in the center still occurs with the same wavelength, but with an inverted phase. This is most likely due to the missing pressure correction, since method 1b shows an inverted phase. Furthermore, the fast error-correcting waves are absent.

To summarize, we could *qualitatively* reproduce the dispersion effects. However, the accuracy and meaning of both the error-correcting waves and the strong oscillation in scenario 3 still pose open questions.

Our analysis so far focused on the value $c = 1000$. To examine the effects with regard to the parameter c, we compare scenario 3 with different values for it, the results being shown in Figure 5.13. For low values of c, the central oscillations begin to form. The shallow water tsunami wave propagates more quickly, and loses in amplitude, the higher c rises. Furthermore, we see that the central oscillations converge towards a certain frequency which does also not change much after reaching $c = 400$. This confirms the statements given in [7]. A dispersive tsunami wave replacing the shallow water tsunami wave can be seen for $c = 600$ and higher. For $c = 1500$, the error-correcting waves propagate even quicker than for $c = 1000$, but the rest of the picture, i.e. the tsunami wave and the dispersion, is almost identical.

A second point we need to rule out is the influence of the mesh size on the central oscillations. To this end, we re-computed scenario 3 with method 2 for subdivision 14 and 16. The result was visually indistinguishable from the solution computed with subdivision 12; hence we also do not show a graphic comparing the three results.

5.3.2 Tsunami Benchmark

We adapted a scenario presented in [18]. In particular, the water height was set to 10 km, and the bathymetry was assumed to be flat in the beginning. For the scenario, we loaded the displacement file using ASAGI from [17]. We ran the simulation using $N_P = 4$, $\gamma = 2$, and we tested both $c = 0$ and $c = 1500$. We enabled AMR and allowed mesh subdivisions between 6 and 12 (both ends included). Once again, we set $a_{\text{ref}} = 0.5$ and $a_{\text{cor}} = 0.8$. See Figure 5.11 for an example during the simulation.

Figure 5.12 shows the results of a simulation with $c = 0$, and a simulation with

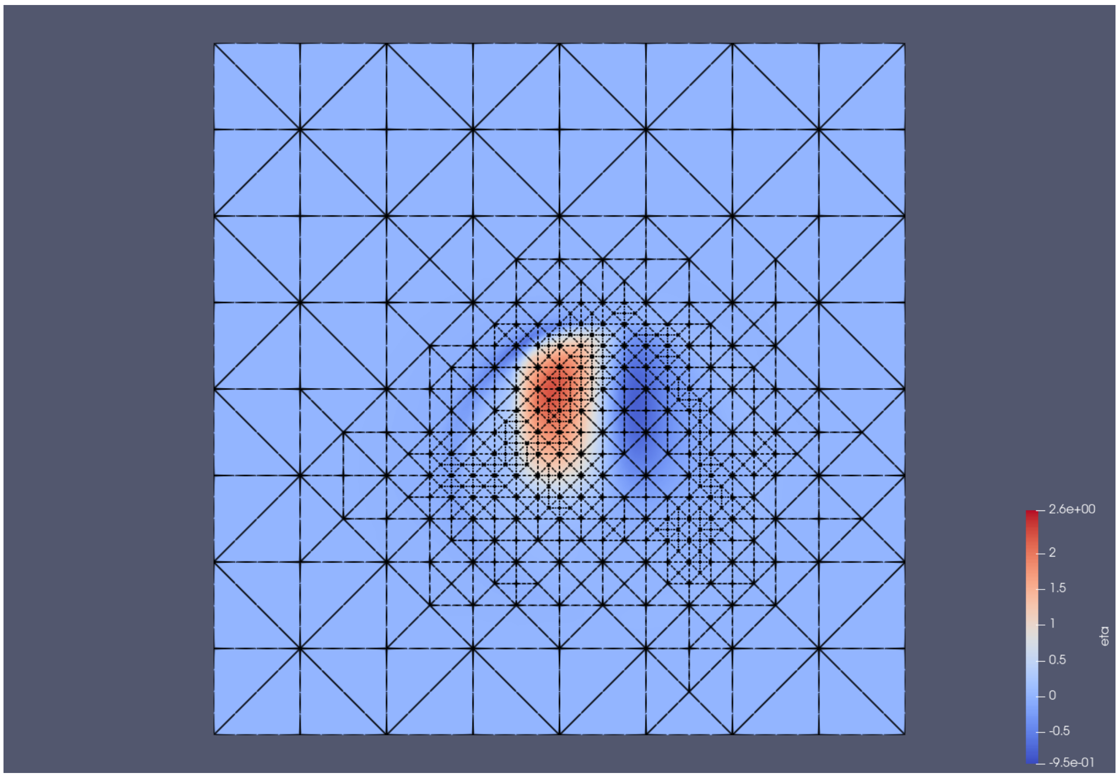
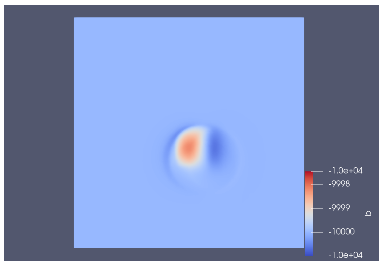
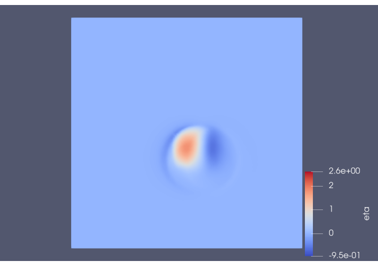


Figure 5.11: A screenshot of the adapted tsunami benchmark simulation from [18] with a sea depth of 10 km. The black lines show the current mesh.

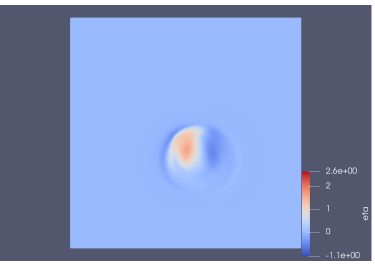
$c = 1500$. We chose the latter value due to $5\sqrt{gh_0} \approx 1566$. The first differences between the two simulations are already clearly visible after about $40s$, and the further the simulation progresses, the more additional waves are visible for the case $c = 1500$.



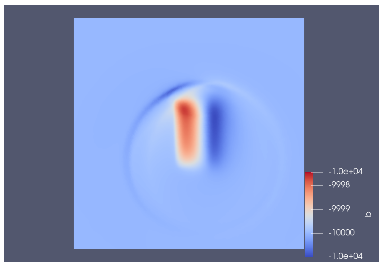
(a) $t = 30s, b$



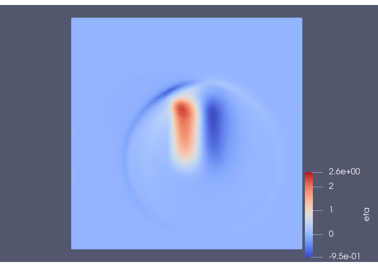
(b) $t = 30s, \eta, c = 0$



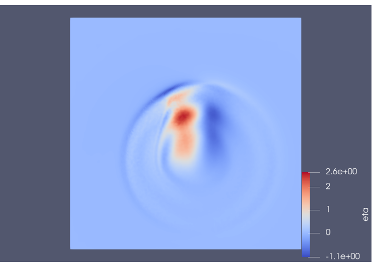
(c) $t = 30s, \eta, c = 1500$



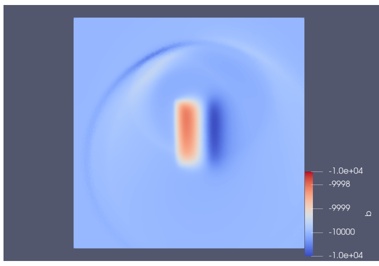
(d) $t = 60s, b$



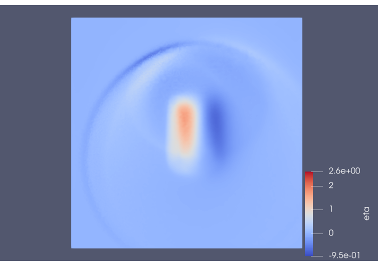
(e) $t = 60s, \eta, c = 0$



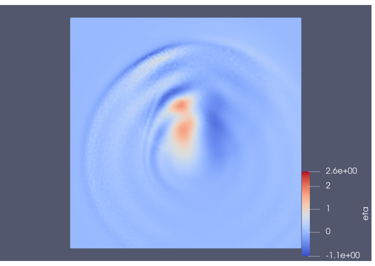
(f) $t = 60s, \eta, c = 1500$



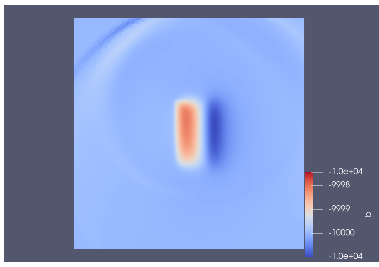
(g) $t = 90s, b$



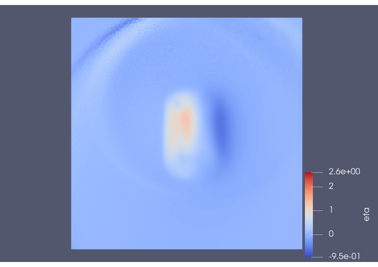
(h) $t = 90s, \eta, c = 0$



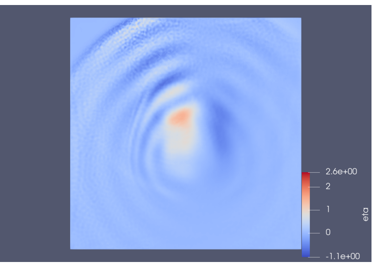
(i) $t = 90s, \eta, c = 1500$



(j) $t = 120s, b$



(k) $t = 120s, \eta, c = 0$



(l) $t = 120s, \eta, c = 1500$

Figure 5.12: An adaption of the tsunami benchmark from [18] with a sea depth of 10 km. We simulate both with $c = 0$ and $c = 1500$, the latter roughly equalling $5\sqrt{gh_0}$. The left-most column shows the bathymetry at the given timesteps, the middle column shows the displacement for $c = 0$, and the right-most column shows the displacement for $c = 1500$.

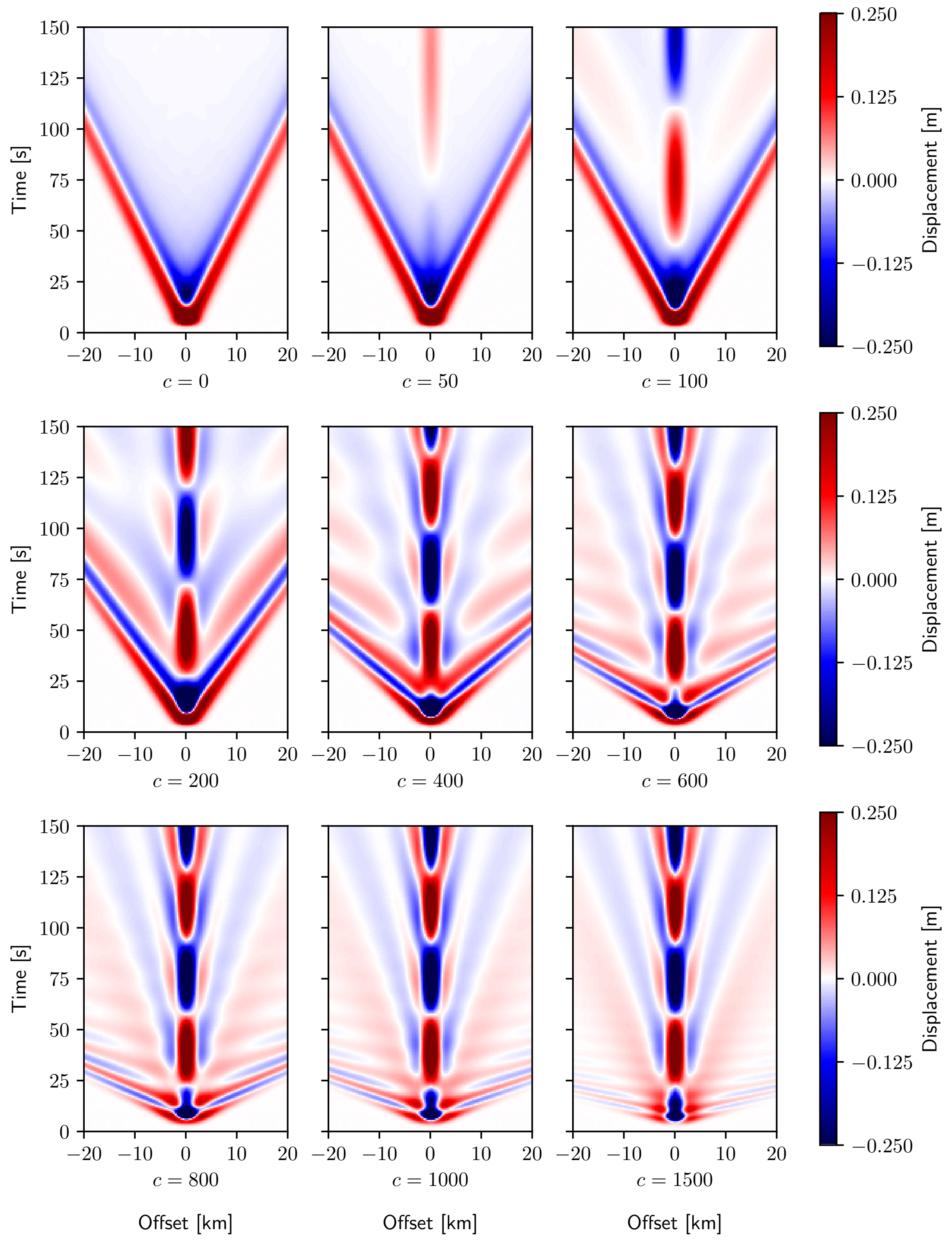


Figure 5.13: Scenario 3 from [1] simulated with method 2 from Table 2.1 and (2.3.8) for different values of c. The polynomial degree was set to $N_P = 4$, and the subdivision to $N_D = 12$. Furthermore, $\gamma = 2$.

6 Outlook

We presented three different ways to simulate tsunamis using the H-BMSS-γ equation system, and we reproduced the scenarios from [1] using them. In addition, we showed the high-order convergence of the H-BMSS-γ model in our implementation. Moreover, we extended our code in sam(oa)^2 to use the Kronecker product and include YaTeTo, so that we can run more complex simulations. Furthermore, we are now able to use adaptive mesh refinement efficiently, and it avoids an integration step in the ADER-DG corrector step by re-using the fixed point found in the predictor.

For future work, all parts offer possibilities to be developed further. From the modelling point of view, it would be interesting to give the artificial wave parameter c a physical relevance. From the numerical side, the next steps are to look into two-dimensional numerical fluxes to be able to simulate with fewer elements, and to further accelerate the predictor iteration. For the ADER-DG simulation in sam(oa)^2, the next step is to generalize the existing code to support arbitrary hyperbolic equations, possibly by merely entering them into a computer algebra system.

References

[1] Lauren S. Abrahams, Lukas Krenz, Eric M. Dunham, Alice-Agnes, Gabriel, and Tatsuhiko Saito. “Comparison of methods for coupled earthquake and 1 tsunami modeling”. In: 2022.

[2] Marie-Odile Bristeau, Anne Mangeney, Jacques Sainte-Marie, and Nicolas Seguin. “An energy-consis-tent depth-averaged Euler system: Derivation and properties”. In: *Discrete & Continuous Dynamical Systems - B* 20.4 (2015), pp. 961–988.

[3] M. J. Castro Díaz and E. Fernández-Nieto. “A Class of Computationally Fast First Order Finite Volume Solvers: PVM Methods”. In: *SIAM Journal on Scientific Computing* 34.4 (2012), A2173–A2196. DOI: 10.1137/100795280. eprint: https://doi.org/10.1137/100795280. URL: https://doi.org/10.1137/100795280.

[4] Michael Dumbser, Dinshaw S. Balsara, Eleuterio F. Toro, and Claus-Dieter Munz. “A unified framework for the construction of one-step finite volume and discontinuous Galerkin schemes on unstructured meshes”. In: *Journal of Computational Physics* 227.18 (2008), pp. 8209–8253. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2008.05.025. URL: https://www.sciencedirect.com/science/article/pii/S0021999108002829.

[5] Michael Dumbser and Eleuterio F. Toro. “A Simple Extension of the Osher Riemann Solver to Non-conservative Hyperbolic Systems”. In: *Journal of Scientific Computing* 48.1 (2011), pp. 70–88. ISSN: 1573-7691. DOI: 10.1007/s10915-010-9400-3.

[6] Michael Dumbser, Olindo Zanotti, Raphaël Loubère, and Steven Diot. “A posteriori subcell limiting of the discontinuous Galerkin finite element method for hyperbolic conservation laws”. In: *Journal of Computational Physics* 278 (2014), pp. 47–75. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2014.08.009. URL: https://www.sciencedirect.com/science/article/pii/S0021999114005555.

[7] C. Escalante, M. Dumbser, and M.J. Castro. “An efficient hyperbolic relaxation system for dispersive non-hydrostatic water waves and its solution with high order discontinuous Galerkin schemes”. In: *Journal of Computational Physics* 394 (2019), pp. 385–416. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2019.05.035. URL: https://www.sciencedirect.com/science/article/pii/S0021999119303730.

[8] C. Escalante, E. D. Fernández-Nieto, T. Morales de Luna, and M. J. Castro. “An Efficient Two-Layer Non-hydrostatic Approach for Dispersive Water Waves”. In: *Journal of Scientific Computing* 79.1 (2019), pp. 273–320. ISSN: 1573-7691. DOI: 10.1007/s10915-018-0849-9.

[9] C. Escalante and T. Morales de Luna. “A General Non-hydrostatic Hyperbolic Formulation for Boussinesq Dispersive Shallow Flows and Its Numerical Approximation”. In: *Journal of Scientific Computing* 83.3 (2020), p. 62. ISSN: 1573-7691. DOI: `10.1007/s10915-020-01244-7`.

[10] Chaulio R. Ferreira and Michael Bader. “A Generic Interface for Godunov-Type Finite Volume Methods on Adaptive Triangular Meshes”. In: *Computational Science – ICCS 2019*. Ed. by João M. F. Rodrigues, Pedro J. S. Cardoso, Jânio Monteiro, Roberto Lam, Valeria V. Krzhizhanovskaya, Michael H. Lees, Jack J. Dongarra, and Peter M.A. Sloot. Cham: Springer International Publishing, 2019, pp. 402–416. ISBN: 978-3-030-22741-8.

[11] SCons Foundation. *SCons: A software construction tool.* URL: `https://scons.org/` (visited on 03/13/2022).

[12] Sergei K. Godunov and I. Bohachevsky. “Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics”. In: *Matematičeskij sbornik* 47(89).3 (1959), pp. 271–306. URL: `https://hal.archives-ouvertes.fr/hal-01620642`.

[13] A. Heinecke, G. Henry, M. Hutchinson, and H. Pabst. “LIBXSMM: Accelerating Small Matrix Multiplications by Runtime Code Generation”. In: *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2016, pp. 981–991. DOI: `10.1109/SC.2016.83`. URL: `https://doi.ieeecomputersociety.org/10.1109/SC.2016.83`.

[14] Inc. Kitware. *CMake.* URL: `https://cmake.org/` (visited on 01/13/2023).

[15] Lukas Krenz, Leonhard Rannabauer, and Michael Bader. “A High-Order Discontinuous Galerkin Solver with Dynamic Adaptive Mesh Refinement to Simulate Cloud Formation Processes”. en. In: *Parallel Processing and Applied Mathematics: 13th International Conference, PPAM 2019*. Lecture Notes in Computer Science 12043. Archive-ID: 2019_11_05_krenz_PPAM_cloudSupercomputing. Springer, Mar. 2020, pp. 311–323. DOI: `10.1007/978-3-030-43229-4_27`. URL: `https://link.springer.com/chapter/10.1007/978-3-030-43229-4_27`.

[16] Lukas Krenz, Carsten Uphoff, Thomas Ulrich, Alice-Agnes Gabriel, Lauren S. Abrahams, Eric M. Dunham, and Michael Bader. “3D Acoustic-Elastic Coupling with Gravity: The Dynamics of the 2018 Palu, Sulawesi Earthquake and Tsunami”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’21. St. Louis, Missouri: Association for Computing Machinery, 2021. ISBN: 9781450384421. DOI: `10.1145/3458817.3476173`. URL: `https://doi.org/10.1145/3458817.3476173`.

[17] Lukas Krenz, Carsten Uphoff, Thomas Ulrich, Alice-Agnes Gabriel, Lauren S. Abrahams, Eric M. Dunham, and Michael Bader. *Supplementary material for 3D Acoustic-Elastic Coupling with Gravity: The Dynamics of the 2018 Palu, Sulawesi Earthquake and Tsunami.* Apr. 2021. DOI: `10.5281/zenodo.5159333`.

[18] E H Madden, M Bader, J Behrens, Y van Dinther, A-A Gabriel, L Rannabauer, T Ulrich, C Uphoff, S Vater, and I van Zelst. “Linked 3-D modelling of megathrust earthquake-tsunami events: from subduction to tsunami run up”. In: *Geophysical Journal International* 224.1 (Oct. 2020), pp. 487–516. ISSN: 0956-540X. DOI: 10.1093/gji/ggaa484. eprint: https://academic.oup.com/gji/article-pdf/224/1/487/34192866/ggaa484.pdf. URL: https://doi.org/10.1093/gji/ggaa484.

[19] Oliver Meister, Kaveh Rahnema, and Michael Bader. “Parallel Memory-Efficient Adaptive Mesh Refinement on Structured Triangular Meshes with Billions of Grid Cells”. In: *ACM Trans. Math. Softw.* 43.3 (Sept. 2016). ISSN: 0098-3500. DOI: 10.1145/2947668. URL: https://doi.org/10.1145/2947668.

[20] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. “SymPy: symbolic computing in Python”. In: *PeerJ Computer Science* 3 (Jan. 2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. URL: https://doi.org/10.7717/peerj-cs.103.

[21] Ninja-Build. *Ninja, a small build system with a focus on speed.* URL: https://ninja-build.org/ (visited on 01/13/2023).

[22] *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications.* New York, NY: Springer New York, 2008. ISBN: 978-0-387-72067-8. DOI: 10.1007/978-0-387-72067-8. URL: https://doi.org/10.1007/978-0-387-72067-8.

[23] Carlos Parés. “Numerical methods for nonconservative hyperbolic systems: a theoretical framework.” In: *SIAM Journal on Numerical Analysis* 44.1 (2006), pp. 300–321. DOI: 10.1137/050628052. eprint: https://doi.org/10.1137/050628052. URL: https://doi.org/10.1137/050628052.

[24] G. Edgar Parker and James S. Sochacki. “Implementing the Picard Iteration”. In: *Neural, Parallel Sci. Comput.* 4.1 (Mar. 1996), pp. 97–112. ISSN: 1061-5369.

[25] Leonhard Rannabauer, Michael Dumbser, and Michael Bader. “ADER-DG with a-posteriori finite-volume limiting to simulate tsunamis in a parallel adaptive mesh refinement framework”. In: *Computers & Fluids* 173 (2018), pp. 299–306. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2018.01.031. URL: https://www.sciencedirect.com/science/article/pii/S0045793018300392.

[26] Sebastian Rettenberger, Oliver Meister, Michael Bader, and Alice-Agnes Gabriel. “ASAGI: A Parallel Server for Adaptive Geoinformation”. In: *Proceedings of the Exascale Applications and Software Conference 2016.* New York, NY: Association for Computing Machinery, 2016.

[27] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version x.y.z).* https://www.sagemath.org. 2022.

[28] Tatsuhiko Saito. *Tsunami Generation and Propagation.* Ed. by Tatsuhiko Saito. Tokyo: Springer Japan, 2019. ISBN: 978-4-431-56850-6. DOI: `10.1007/978-4-431-56850-6`.

[29] David Schneller. "Discontinuous Galerkin Schemes for Dispersive Non-Hydrostatic Shallow Water Equations". Mar. 15, 2022.

[30] The sam(oa)^2 Team. *sam(oa)^2.* URL: `https://gitlab.lrz.de/samoa/samoa` (visited on 03/02/2022).

[31] The SeisSol Team. *SeisSol source code repository.* URL: `https://github.com/SeisSol/SeisSol` (visited on 01/14/2023).

[32] Gerald Teschl. *Ordinary Differential Equations and Dynamical Systems.* Graduate Studies in Mathematics 140. 2012. ISBN: 978-0-8218-8328-0.

[33] Carsten Uphoff and Michael Bader. "Yet Another Tensor Toolbox for discontinuous Galerkin methods and other applications". In: *CoRR* abs/1903.11521 (2019). arXiv: `1903.11521`. URL: `http://arxiv.org/abs/1903.11521`.

[34] Olindo Zanotti, Francesco Fambri, Michael Dumbser, and Arturo Hidalgo. "Space–time adaptive ADER discontinuous Galerkin finite element schemes with a posteriori sub-cell finite volume limiting". In: *Computers & Fluids* 118 (Sept. 2015), pp. 204–224. ISSN: 0045-7930. DOI: `10.1016/j.compfluid.2015.06.020`. URL: `http://dx.doi.org/10.1016/j.compfluid.2015.06.020`.

List of Figures

List of Tables

List of Algorithms