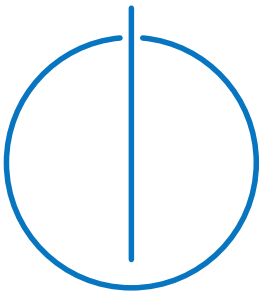
Technical University of Munich
Department of Informatics

Bachelor's Thesis in Informatics

Grid Projection for Simulations of Tsunami-Genesis

Maximilian Schmeller



Technical University of Munich
Department of Informatics

Bachelor's Thesis in Informatics

Grid Projection for Simulations of Tsunami-Genesis

Gitterprojektionen zu Simulationen der Entstehung von Tsunamis

Maximilian Schmeller

Supervisor:	Univ.-Prof. Dr. Michael Bader
Advisors:	M. Sc. Lukas Krenz,
	M. Sc. Leonhard Rannabauer
Submission date:	2020-08-15

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

Munich, July 16, 2020

Abstract

As CPU time is a valuable and limited resource in high performance computing it is desirable to use it as efficiently as possible. This often means reducing I/O operations and outputting less data. This also holds true for the seismic wave propagation simulator SeisSol and we investigate whether outputs such as time-dependent seafloor displacements, water velocities or sea surface elevation can be omitted without negatively affecting subsequent tsunami simulations using this data.

To be able to evaluate the mentioned outputs as tsunami simulation inputs, the unstructured tetrahedral and triangular meshes (= simplex meshes) produced by SeisSol have to be rasterized into a regular grid of rectangular cells. This is done by sampling the SeisSol meshes at each of the regular grid's cell centers. The variables in the tetrahedral mesh are averaged across the water height while those in the triangular meshes are rasterized directly. To check whether a cell center is inside a simplex we develop a criterion using the Hesse Normal Form for each of the faces/edges of the simplex.

With the rasterized data we find that both initial velocities and surface elevations can be omitted as tsunami input data for scenarios with a flat seafloor, resulting in a difference in sea surface height of up to a few centimeters compared to using complete initial conditions. The surface elevation can be approximated using Kajiura's Filter which results in less high-frequency waves compared to copying the seafloor displacements to the surface. This filter does not have a significant effect on Shallow Water Equation simulations since those do not simulate frequency dispersion and the removal of high frequencies in the wave profile does therefore not affect the simulation much.

Notation

$\mathbf{v}$	$\triangleq$	$\begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}$	An n-dimensional column vector
$\langle \mathbf{r}, \mathbf{s} \rangle$	$\triangleq$	$\sum_{i=1}^{n} r_i s_i$	The standard dot product
$\|\mathbf{v}\|$	$\triangleq$	$\sqrt{\langle \mathbf{v}, \mathbf{v} \rangle}$	The $L2$-norm of a vector
D_{xy}	$\triangleq$	$\{(x, y) \mid (x, y, z) \in D\}$	Only x- and y-components of the points in set D

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgement

I would like to thank my advisors Leonhard Rannabauer and Lukas Krenz for their great support throughout the writing of this thesis. Their time and advice have been very valuable and it has been a joy to work with them.

Furthermore, I would like to thank my friends Julien Kollmann, Lukas Mautner and Markus Schacherbauer, not only for proof-reading this thesis but also for enduring my rants about Covid-19 and hackers of supercomputers.

I would also like to express my appreciation for the Leibniz Supercomputing Centre, the Technical University of Munich and the German system of education that all enabled me to use the resources needed for this thesis entirely free of charge.

Chapter 1

Introduction

Gaining a better understanding of tsunamis and tsunamigenic earthquakes has proven difficult many times as computational resources are still valuable and limited to this date. Nevertheless, modern supercomputers can simulate such seismic events over large domains in full 3D and with high physical accuracy [17].

As much as the 3D outputs of such simulations promise to be the perfect starting point for tsunami simulators such as Shallow-Water-Equation (SWE) solvers, it is desirable to minimize the amount of output data in order to maximize the resources available for the earthquake simulation [18, sec. 6.3]. Additionally, the full simulation of tsunami genesis requires manual preparation and data which might not always be available, such as the sea-surface elevation, water velocities and the seafloor displacement over time.

Consequently, the question whether SWE solvers can work with incomplete input data without any noticeable loss in precision and accuracy of their results arises.

To answer this question, an efficient method to project the unstructured output meshes of the seismic wave propagation simulator SeisSol (`http://www.seissol.org`) onto structured 2D meshes, suitable for the SWE solver sam(oa)^2 (`https://gitlab.lrz.de/samoa/samoa`), is developed in chapter 2. Using this method, different subsets of the data are extracted from the original SeisSol outputs and algorithms to augment incomplete sets of data are explored.

Using a full snapshot of a tsunami after an earthquake as the baseline, we investigate the effects of including or excluding initial velocities in the water body and the effects of time-dependent versus static seafloor displacement in chapter 3.

For cases where only the seafloor displacements are known, Kajiura's Filter can be applied to seafloor displacements to better approximate a

realistic water surface elevation instead of copying the seafloor elevation to the surface 1:1. This is discussed and evaluated in chapter 4.

Chapter 2

Projection of Non-uniform Simplex Meshes onto 2D Uniform Rectangular Grids

Definition 2.1 (Rasterization) *In this thesis: The process of discretizing a grid of polygons or polytopes into a raster consisting of rectangles or cubes.*

While the generation of tetrahedral meshes is a well-studied problem [20], the rasterization of such meshes into uniform rectangular grids receives little academic interest outside of computer graphics. Algorithms for rendering unstructured meshes can be found in abundance, but the vast majority of them require massively parallel architectures such as GPUs [5, 6]. Nevertheless, the basic principles of tetrahedron rasterization apply regardless of architecture. There are two approaches for rasterizing simplex meshes: *tile-order* and *simplex-order*. The former means that iterating over each cell of the desired raster, overlapping simplices are determined. The latter approach iterates through the simplices in the unstructured grid and determines the overlapping cells of the raster. While both approaches are valid, one can vastly outperform the other in certain applications and vice-versa, due to their different memory access patterns. This is discussed in-depth in section 2.2.1.

For evaluation purposes, *Scenario A* from [13] is used and is from now on referred to as "the ASCETE scenario/test case". It features a domain of $750\,\mathrm{km} \times 750\,\mathrm{km}$, a flat seafloor at depth $-2\,\mathrm{km}$ and a planar fault near the middle of the domain which dips at 16° and intersects the seafloor surface. The earthquake features maximum and minimum vertical seafloor displacements of $1.6\,\mathrm{m}$ and $-1.0\,\mathrm{m}$ respectively. The simulated timeframe is $300\,\mathrm{s}$, beginning with the start of the earthquake. This earthquake has

been simulated with SeisSol and the output mesh features 4.6 million tetrahedra and 5.5 million additional triangles, and thus can serve as a realistic benchmark for the algorithms and approaches described in this thesis.

Furthermore, since the seafloor of this model is flat, effects introduced by uneven bathymetry are ruled out.

2.1 Mathematical Principles

Definition 2.2 ((*n*-)Simplex) *An (n-)simplex σ is the set of points in $\mathbb{R}^n$ spanned by the points in a geometrically independent set $\mathcal{S} = \{a_0, \ldots, a_n\}$ of $n+1$ vertices [15, p. 2ff]. More intuitively, the vectors $a_1 - a_0, \ldots, a_n - a_0$ are linearly independent [15, p. 3].*

For this thesis, it is entirely sufficient to know that triangles and tetrahedra are 2-simplices and 3-simplices respectively and unless otherwise noted, the term "simplex" is used to exclusively refer to triangles and tetrahedra in the rest of the thesis. This abstraction will prove to be beneficial when solving the problems in the upcoming sections.

2.1.1 Grid Structures

To align with common terminology, the naming schemes from [24] are used throughout the thesis.

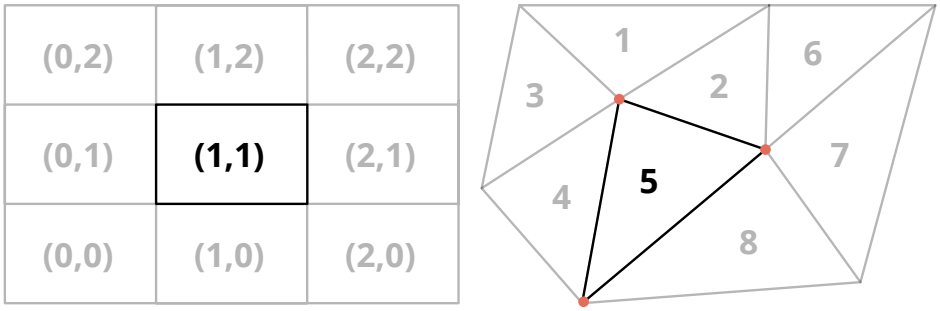
In the following sections, only *regular grids* and *unstructured simplex grids* (consisting entirely of either triangles or tetrahedra; called only *unstructured grids* from now on) are of relevance.

Every grid has a rectangular (2D) or cuboid (3D) domain

$$D = D_x \times D_y \times D_z = [x_{min}; x_{max}] \times [y_{min}; y_{max}] \times [z_{min}; z_{max}] \,.$$

Furthermore, regular grids $G = (D, \Delta x, \Delta y, \Delta z)$ have a cell size of $\Delta x \times \Delta y \times \Delta z$ with cells being indexed by indices i, j, k and $(x_{max} - x_{min})$ being divisible by Δx (analogous for y and z). The domain occupied by cell (i, j, k) is therefore

$$\begin{aligned} S_G(i,j,k) = &\quad [x_G(i); x_G(i+1)] \times [y_G(j); y_G(j+1)] \times [z_G(k); z_G(k+1)] \\ := &\quad [x_{min} + i\Delta x; x_{min} + (i+1)\Delta x] \\ &\times [y_{min} + j\Delta y; y_{min} + (j+1)\Delta y] \\ &\times [z_{min} + k\Delta z; z_{min} + (k+1)\Delta z] \,. \end{aligned}$$



(a) Regular grid (b) Unstructured grid

Figure 2.1: Indices of cells in a regular grid and an unstructured one. In the unstructured grid, the vertices $V(\sigma_5)$ of the black simplex are highlighted in orange. Both grids are 2D for simplicity.

Unstructured grids $H = (D, \Sigma)$ are indexed with a single index i with each simplex $\sigma_i \in \Sigma$ having an explicit set of vertices $V(\sigma_i)$ associated with it, with Σ being the set of all simplices in H (fig. 2.1).

Both types of grids are required to cover the entirety of D_{xy}. This is necessary to ensure that the rasterization approaches in later sections can rely on finding overlapping simplices for each cell of the regular grid.

While the details of how unstructured grids store data can vary, this thesis focuses on the format of the grids output by SeisSol:
The three unstructured output grids all occupy the same domain in x- and y-direction with each of the grids only occupying a part of their combined z-domain. Furthermore, the variables stored for each simplex cell in these meshes are evaluated at the respective cell center. The first mesh ("**volume**") is tetrahedral and represents the body of water and the volume of ground surrounding it. The second ("**surface**") and third one ("**floor**") are triangular meshes, the former representing the sea surface when at rest and the latter representing the seafloor. All three meshes are shown in fig. 2.2.

2.1.2 Approaches to the Point-in-Simplex Problem

Instead of trying to compute the exact volume of the intersection between a tetrahedron and a regular grid cell, which is more compute-intensive, we simply approximate the intersection by sampling the simplex at the center of the regular grid cell. With the regular grid cells becoming small in comparison to the simplices in the unstructured grid, the rasterization error is minimized as shown in section 2.1.4.

The point-in-simplex problem is well-studied and two approaches are

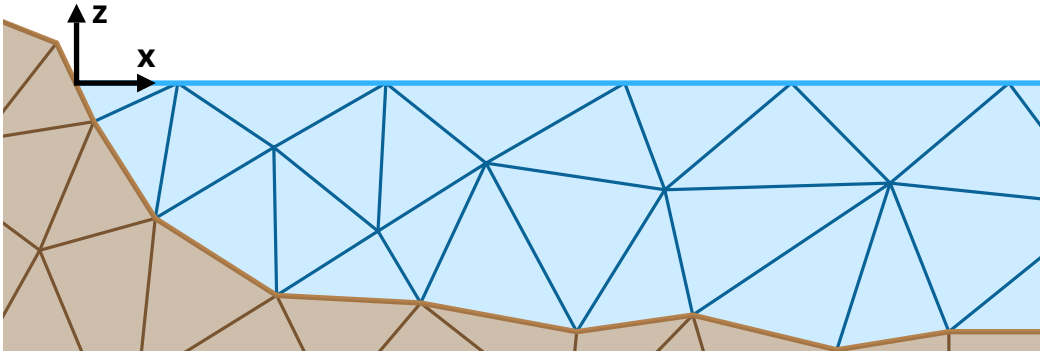


Figure 2.2: SeisSol output meshes (planar slice). **Volume** shown in dark blue/brown with light blue/brown fill, **surface** in light blue (thick) and **floor** in light brown (thick).

particularly common:

- Barycentric Coordinates
- Hesse Normal Form

Barycentric Coordinates

Summarizing [9, sec. 6.1.1], barycentric coordinates are a way to express a point $\mathbf{p}$ as a combination of a triangle's vertices $\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2}$:

$$\mathbf{p} = \sum_{i=0}^{2} a_i \mathbf{p_i} \text{ with } a_0 + a_1 + a_2 = 1\,. \tag{2.1}$$

If

$$\forall i \in \{0,1,2\}\ (a_i \geq 0) \tag{2.2}$$

holds, then $\mathbf{p}$ is inside the triangle.

To obtain the coefficients a_i, we can solve the following system of linear equations consisting of the above formulae

$$\begin{pmatrix} p_{0,x} & p_{1,x} & p_{2,x} \\ p_{0,y} & p_{1,y} & p_{2,y} \\ p_{0,z} & p_{1,z} & p_{2,z} \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}\,. \tag{2.3}$$

This approach can be extended to tetrahedra [26, sec. 11.7, sec. 11.9] which results in the general formulation

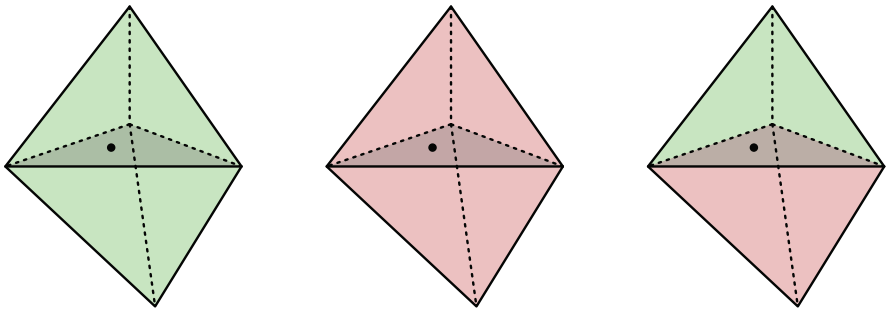
$$\mathbf{p} = \sum_{i=0}^{n} a_i \mathbf{p_i}\,, \sum_{i=0}^{n} a_i = 1;\ n \in \{2,3\} \tag{2.4}$$

and if

$$\forall i \in \{0, \ldots, n\}.\ a_i \geq 0 \tag{2.5}$$

holds, then $\mathbf{p}$ is inside of the simplex with vertices $\mathbf{p_0}, \ldots, \mathbf{p_n}$.

Note that the *inside* of a simplex also includes faces, edges and vertices when using eq. (2.5) (=: inclusive). This, however, leads to a problem: In non-uniform grids, simplices have neighbors and thus points can be *inside* multiple simplices according to the above definition. This is an unwanted property as special care would need to be taken to correctly average the values of multiple simplices when rasterizing. A possible remedy would be to only consider points inside the simplex where all $a_i > 0$ (=: exclusive). This excludes points on the faces, edges and vertices of the simplex but leaves these points excluded from all simplices which causes the problem that these points cannot be associated with any value during rasterization. Figure 2.3 compares these approaches with the Hesse Normal Form approach discussed next.



(a) BC, inclusive (b) BC, exclusive (c) HNF

Figure 2.3: A point lying on the shared face of two neighboring tetrahedra. The point is considered to be *inside* of green tetrahedra and outside of red ones. *BC* is short for barycentric coordinates.

Hesse Normal Form

According to [7, p. 540ff], a plane E in $\mathbb{R}^3$ can be represented in Hesse Normal Form (HNF) as

$$\langle \mathbf{n}, \mathbf{x} \rangle = -p\,, \tag{2.6}$$

with $\mathbf{n}$ being a normal vector of E and

$$\|\mathbf{n}\| = 1\,. \tag{2.7}$$

p is a constant and all points $\mathbf{x}$ satisfying eq. (2.6) lie in E.

Given three distinct points $\mathbf{r}, \mathbf{s}, \mathbf{t}$ in E, its HNF can be obtained as follows:

$$\mathbf{n}' = (\mathbf{s} - \mathbf{r}) \times (\mathbf{t} - \mathbf{r}) \,, \tag{2.8}$$

$$\mathbf{n} = \frac{\mathbf{n}'}{\|\mathbf{n}'\|} \,, \tag{2.9}$$

$$p = -\langle \mathbf{n}, \mathbf{r} \rangle \,. \tag{2.10}$$

In eq. (2.8), a normal vector of E is computed as the cross product of two vectors that span E. Due to the result of the cross product being perpendicular to its (linearly independent, non-zero) arguments, $\mathbf{n}'$ is perpendicular to the plane as a whole and thus a normal vector of E. Through normalization of $\mathbf{n}'$ in eq. (2.9), $\mathbf{n}$ is obtained as is required by eq. (2.7). To calculate the constant p, an arbitrary point in E (the choice here being $\mathbf{r}$) is inserted into eq. (2.6).

With E in HNF, the *signed distance* of an arbitrary point $\mathbf{q}$ to the plane can now be calculated as shown in [7, p. 541]:

$$d = \langle \mathbf{n}, \mathbf{q} \rangle + p \,. \tag{2.11}$$

The last observations missing before we can solve the point-in-simplex problem are:

- The point $\mathbf{m} = \frac{1}{n+1} \sum_{i=0}^{n} \mathbf{p_i}$ always lies within (not on the edges or faces) the corresponding n-simplex.
- Two points that have equal-signed distances to E lie on the same side of E.

Given a tetrahedron σ with its set of vertices $V(\sigma) = \{\mathbf{p_0}, \ldots, \mathbf{p_3}\}$, the four planes $E_0, \ldots, E_3$ can be defined with E_i being the plane spanned by $V(\sigma) \setminus \{\mathbf{p_i}\}$ for each i. Each of the four planes' normal vectors $\mathbf{n_i}$ is now chosen such that the signed distance of $\mathbf{m}$ is positive with respect to E_i:

$$\langle \mathbf{n_i}, \mathbf{m} \rangle + p_i \geq 0 \,. \tag{2.12}$$

Graphically, this means that all of the normal vectors are pointing from the faces of the tetrahedron to its inside. Points having positive distances $d_i > 0$ to each of the planes E_i lie on the positive half-space [7, p. 541] of each plane and are therefore inside of the tetrahedron. A distance of $d_i = 0$ means that the point lies on the plane E_i. This leads to the same problem the approach using barycentric coordinates left us at where we have to decide in which of

two neighboring simplices a point lies when it is on a plane. However, in contrast to the barycentric approach, the HNF approach allows us to define a criterion which assigns points on faces to single distinct tetrahedra using the direction of $\mathbf{n_i}$.

It is trivial that for two non-overlapping tetrahedra that share a common face spanning a plane F, the respective normal vectors $\mathbf{n_1}$ and $\mathbf{n_2}$ pointing to the inside of their tetrahedron have to be the exact opposite of each other:

$$\mathbf{n_1} = -\mathbf{n_2}\,. \tag{2.13}$$

Both are perpendicular to F, have magnitude 1 and point to different half-spaces of F. This fact allows the following inclusion/exclusion rule: Let $\preceq$ be a total order of all unit vectors in $\mathbb{R}^3$ which shall be defined as

$$\begin{aligned}\mathbf{u} \preceq \mathbf{v} := \quad & (u_1 \leq v_1) \\ & \vee\, (u_1 = v_1) \wedge (u_2 \leq v_2) \\ & \vee\, (u_1 = v_1) \wedge (u_2 = v_2) \wedge (u_3 \leq v_3)\,.\end{aligned} \tag{2.14}$$

From eq. (2.13), eq. (2.14) and $\mathbf{n_1} \neq \mathbf{0}$ follows that

$$\mathbf{n_1} \prec \mathbf{n_2} \oplus \mathbf{n_2} \prec \mathbf{n_1}\,, \tag{2.15}$$

with $\oplus$ being the exclusive-or operator, must always hold.

When given a point in a tetrahedron σ which is lying on one of its faces (edges), it is considered to be *inside* σ if and only if $\mathbf{n_i} \prec -\mathbf{n_i}$ holds. $\mathbf{n_i}$ is the normal vector of the face as described in eq. (2.12).

The error introduced by choosing one tetrahedron over another when a point lies exactly between them is minimal since the variables stored in those tetrahedra are typically samples of a (mostly) continuous function and therefore do not differ significantly in value. For the case that they do, the points lying exactly between tetrahedra are still rare compared to those lying on the insides and thus do not affect the end result significantly.

Using the HNF for lines instead of planes [7, p. 290] this method can be applied to triangles analogously.

2.1.3 Sampling Simplex Grids

The point-in-simplex check is the first step in rasterization as it determines which simplices to sample into a regular grid cell. The exact method of calculating the rasterized value the cell receives is discussed now. To facilitate the following explanations, the functions in table 2.1 are introduced.

Table 2.1: Utility functions for working with simplices.

Definition	Solution
$tetAt_H(x,y,z) = \sigma \in H_\Sigma \mid (x,y,z) \in \sigma$	HNF
$triAt_H(x,y) = \sigma \in H_\Sigma \mid \exists z \in \mathbb{R}\ ((x,y,z) \in \sigma)$	HNF
$q(\sigma) \triangleq$ The value of variable q in simplex σ	given
$zAt_H(x,y) = z \in \mathbb{R} \mid (x,y,z) \in triAt_H(x,y)$	to be solved

Table 2.2: The three methods needed for rasterizing the input grids. Only one method is needed per rasterization of a grid and variable.

Method Name	Mathematical Representation	Description
Standard rasterization	$q_{x,y} = q(triAt_H(x,y))$	The value of variable q in the triangle at point $(x,y)^T$
Depth rasterization	$h_{x,y} = -zAt_H(x,y)$	The depth, extracted from the grid's geometry, at point $(x,y)^T$
Depth averaging	$q_{x,y;avg} = \frac{1}{h_{x,y}} \int_{-h_{x,y}}^{0} q(tetAt_H(x,y,z))dz$	The average of a variable q over a water column of height $h_{x,y}$

Recalling the three SeisSol meshes **volume**, **surface** and **floor** from section 2.1.1, the process of mapping the variables the meshes provide to those desired as outputs is discussed next. In total, three different methods are required to rasterize all variables of an unstructured grid H. Those methods are listed in table 2.2.
The outputs needed to initialize all degrees of freedom of an SWE simulation (see chapter 3 for details) are the bathymetry height b, water height h and water momentum $\mathbf{p}$. All of these can be derived from the variables we extract: b, vertical seafloor displacement d, sea surface displacement η and the water velocity vector $\mathbf{v}$. The conversion of these variables to the ones stated above is left to the tsunami solver. In table 2.3, the mappings between in- and output variables and rasterization methods used are listed.
Generally speaking, *depth averaging* is only needed when rasterizing tetrahedra into a 2D grid and *depth rasterization* is required when extracting geometric data from triangles. For rasterizing variables from triangles, *standard rasterization* is used.

Table 2.3: The mappings from inputs to outputs with the respective rasterization methods.

Output variable	Inputs	Rasterization method
b	**floor**: mesh geometry	Depth rasterization
d	**floor**: W	Standard rasterization
η	**surface**: W	Standard rasterization
$\mathbf{v} = (u, v)^T$	**volume**: u, v	Depth averaging

Depth-averaging

The depth averaging method above is a path integral over multiple tetrahedra and as such, is not computable without further modifications. Therefore, with $H = (D, \Sigma)$ and $G = (D, \Delta x, \Delta y, \Delta z)$, we approximate this integral as a sum of samples of H at G's cell centers:

$$\tilde{q}_{x,y;avg} = \frac{1}{|C|} \sum_{z \in C} q(tetAt_H(x, y, z)) \,, \tag{2.16}$$

with C being the set of z-coordinates of cell centers above the bathymetry height:

$$C = \{z \in \mathbb{R} \mid \exists k \in \mathbb{N}_0 \ \left((z = z_{min} + (k + \frac{1}{2})\Delta z) \wedge (z \geq -h_{x,y}) \right) \} \,. \tag{2.17}$$

The error introduced by this simplification is discussed in section 2.1.4.

Depth-rasterization

This method extracts the water depth at a certain point $\mathbf{p} = (x, y)$ from the geometry of the unstructured seafloor mesh. To do that, the triangle σ that overlaps $\mathbf{p}$ has to be determined. Then the exact depth at $\mathbf{p}$ can be found by interpolating between the depths at σ's vertices.

Here, the bathymetric coordinates discussed in section 2.1.2 can be utilized. As explained in [26, ch. 11.4], these coordinates can be used for interpolating between values at the triangle's vertices within said triangle. This is done using the three barycentric coordinates a_0, a_1, a_2 of $\mathbf{p}$ which correspond to σ's vertices $\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2}$. The interpolated z-coordinate at $\mathbf{p}$ is then

$$zAt(x, y) = a_0 \mathbf{p_0}_z + a_1 \mathbf{p_1}_z + a_2 \mathbf{p_2}_z \,. \tag{2.18}$$

2.1.4 Error Analysis

By sampling a unstructured grid H only at the cell centers of a regular grid $G = (D, \Delta x, \Delta y, \Delta z)$ we introduce a numerical error which reduces with the cell size of G. The exact value of a cell (i, j) in G after rasterization of a tetrahedral grid is

$$q_{i,j;avg} = \frac{1}{\Delta x \Delta y} \int_{x_G(i)}^{x_G(i+1)} \int_{y_G(j)}^{y_G(j+1)} q_{x,y;avg} \; dy \; dx \tag{2.19}$$

and analogously for a triangle grid

$$q_{i,j} = \frac{1}{\Delta x \Delta y} \int_{x_G(i)}^{x_G(i+1)} \int_{y_G(j)}^{y_G(j+1)} q_{x,y} \; dy \; dx \,. \tag{2.20}$$

The method of sampling at cell centers delivers

$$\tilde{q}_{i,j;avg} = \tilde{q}_{x_G(i+\frac{1}{2}),y_G(j+\frac{1}{2});avg} \tag{2.21}$$

for tetrahedra and

$$\tilde{q}_{i,j} = q_{x_G(i+\frac{1}{2}),y_G(j+\frac{1}{2})} \tag{2.22}$$

for triangles. The relative error made by point sampling for a given cell of G is then

$$f_{i,j;rel;tet} = \frac{q_{i,j;avg} - \tilde{q}_{i,j;avg}}{q_{i,j;avg}} \tag{2.23}$$

and

$$f_{i,j;rel;tri} = \frac{q_{i,j} - \tilde{q}_{i,j}}{q_{i,j}} \tag{2.24}$$

respectively. All further relative errors in this thesis are defined analogously.

To find out which $\Delta x, \Delta y, \Delta z$ are suitable for a given simplex size, we devise the following scenario: With domain $D = [0; 1]^3$ and a tetrahedron σ with $V(\sigma) = \{(0,0,0)^T, (1,1,0)^T, (1,0,1)^T, (0,1,1)^T\}$, the unstructured grid $H = (D, \{\sigma, \sigma_{001}, \sigma_{010}, \sigma_{100}, \sigma_{111}\})$ and the function $G(\delta) = \left(D, \frac{1}{\delta}, \frac{1}{\delta}, \frac{1}{\delta}\right)$ producing regular grids at different granularities δ are defined. $\sigma_{001}, \sigma_{010}, \sigma_{100}, \sigma_{111}$ are

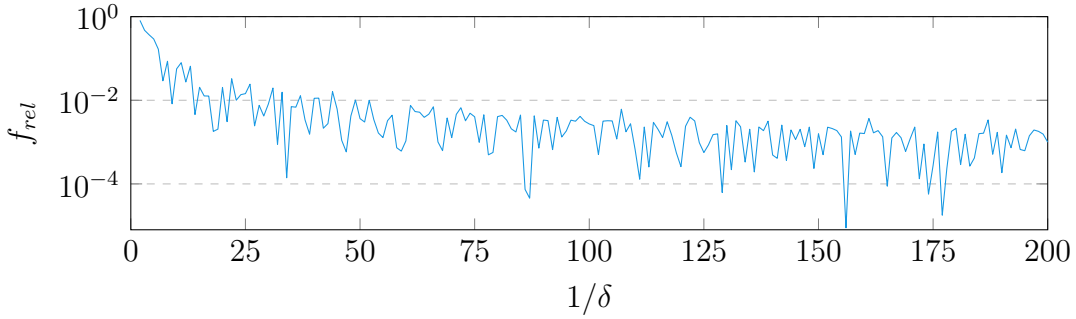


Figure 2.4: The relative error to the volume of the tetrahedron described in section 2.1.4 when sampled at $(\frac{1}{\delta})^3$ cell centers.

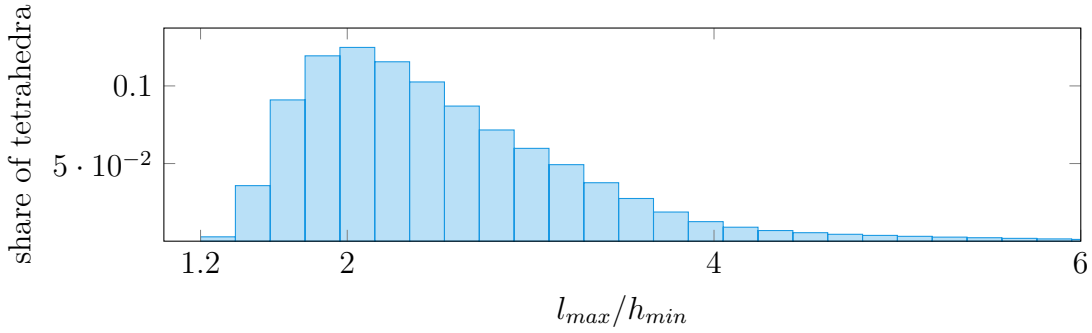


Figure 2.5: Distribution of tetrahedron aspect ratios in the ASCETE test case.

the four tetrahedra at the cubic domain's unoccupied corners which are chosen such that the domain is completely filled. In the cell σ, the value 1 is stored while all other cells are set to 0. This way, when integrating over the domain, the expected value is $\frac{1}{3}$, which is the volume of σ.

The relative errors observed for $\delta \in \{1/s \mid s \in \mathbb{N} \wedge 2 \leq s \leq 200\}$ are shown in fig. 2.4. This scenario does not cover all possible simplices and particularly disproportionate ones can have much larger sampling errors, e.g. ones that are so thin in one dimension that they lie completely inbetween cell centers of a given regular grid and are thus never sampled. However, such simplices rarely exist in meshes that are used for scientific simulations since they negatively affect simulation quality [20, ch. 1]. Using the definition of *aspect ratio* from [20, sec. 9.1] as a measure for mesh quality we can confirm that high-aspect-ratio (read: low-quality) tetrahedra occur very rarely in real meshes (Figure 2.5).

To achieve a relative error of 10^{-2} on low-aspect-ratio (high-quality)

tetrahedra, approximately 50^3 samples have to be taken for each tetrahedron, resulting in $5.8{\cdot}10^{11}$ total samples in a scenario like the ASCETE test scenario which features $\approx 4.6 \cdot 10^6$ tetrahedra.

Additionally, a common property among input grids can reduce the error further: The functions of which the values are stored in the simplex cells typically have a low gradient (except for areas near the fault), i.e. their values differ only slightly between neighboring cells. Thus, when the sampling error for an individual simplex is large, the sampled value remains accurate as the whole neighbourhood of the simplex is similar in value. As discucssed in section 2.1.2, a point sample always lies inside *exactly one* simplex.

To illustrate this, the tetrahedral cells of the ASCETE scenario are sampled at $t = 80\,\mathrm{s}$ with $\Delta x = \Delta y = \Delta z = 100\,\mathrm{m}$ and the relative error between the variables in the rasterized grid and the input grid is calculated after integrating them over the whole domain. This yields $f_{rel;3d} \approx 1.30\,\% = 1.3 \cdot 10^{-2}$ and $f_{rel;2d} \approx 0.17\,\% = 1.7 \cdot 10^{-3}$ for the 3D volume grid and the 2D floor/surface grids respectively. The number of samples was on average 46.8^3 per tetrahedron.

2.2 Projection Algorithm and Data Structures

The problem of grid rasterization as described in the previous sections lends itself well to massive parallelization, like it is found in graphics cards. However, the size of the inputs and outputs of the algorithm (hundreds of gigabytes in typical scenarios) and the fact that the program can benefit from large amounts of RAM, have lead to developing this algorithm for CPU clusters instead. As later discussed in section 2.3, the systems optimized for are the CoolMUC2[1] and CoolMUC3[2] clusters at the Leibniz Supercomputing Centre (LRZ).

The current implementation fully utilizes multithreading capabilities of a single compute node but does not feature multi-node support. This is due to the language chosen for implementation being *Julia*[3] which, at the time of writing, has no official support from the Leibniz Supercomputing Centre[4] for running on multiple nodes in parallel.

Also, the conscious choice has been made to support running the algorithm on personal computers to facilitate development and to allow for fast execution on small synthetic scenarios.

[1] `https://doku.lrz.de/display/PUBLIC/CoolMUC-2`
[2] `https://doku.lrz.de/display/PUBLIC/CoolMUC-3`
[3] `https://julialang.org/`
[4] `https://doku.lrz.de/display/PUBLIC/High+Performance+Computing`

Table 2.4: The binary input files to the program as referenced by the XDMF header.

Filename	File Contents
`geometry.bin`	The corner points of all simplices in the unstructured grid
`topology.bin`	The tuples of 3/4 corner point IDs that define the simplices
`<varname>.bin`	The numeric values of variable `<varname>`, the ordering being $(timestep, simplexID)$[7]

To receive a copy of the code refer to section A.1.

2.2.1 Rasterization Order

As already mentioned in chapter 2, the algorithm can be designed in two different ways:

- Operating in tile-order, i.e., iterating over the regular grid cells and finding the overlapping simplices for each one.
- Operating in simplex-order, iterating over the unstructured grid's simplices and finding the overlapping regular grid cells.

To decide which of these approaches is appropriate for our algorithm, we have to take a look at the format of the input files: All inputs are given in the XDMF format[5] which features an XML header file. This header points to a collection of binary files of which the ones shown in table 2.4 are of interest. The outputs of the algorithm are written in NetCDF format[6], with a $(timestep, y, x)$[7] ordering.

This means that, for accessing the filesystem optimally, i.e., sequentially, we need to *read* in simplex-order and *write* in tile-order.

This is achieved by rasterizing the unstructured grids in *simplex-order* to a regular grid kept *in memory*. When all simplices of a grid have been rasterized, the in-memory regular grid is then written to the filesystem sequentially.

Alternatively, one could read the topology of the unstructured grid into a data structure like a quad- or octree and then rasterize in tile-order. However, this has proven to be slower and more memory-intensive than the former method.

[5] `http://xdmf.org/index.php/Main_Page`

[6] `https://www.unidata.ucar.edu/software/netcdf/`

[7] Orderings given as column-major; The right indices iterate faster than the left ones.

2.2.2 The Algorithm

The abstraction *simplex* made in section 2.1 now allows us to develop *one* algorithm for the tetrahedra and triangles, with only minor exceptions.

The grids of $gridType \in \{\textbf{floor}, \textbf{surface}, \textbf{volume}\}$ are rasterized after one another to the same regular grid G.

Algorithm 1 first allocates memory for the cells of grid G to accomodate each of the rasterized variables. The cells in $grid_{out;vars}$ will receive the values in the rasterized grid and $grid_{out;cnt}$ counts the simplices that overlap a particular cell when projected onto the xy-plane. The latter array is only needed for the tetrahedral **volume** grid in order to build a running average.

Algorithm 1 The grid rasterization algorithm

1: **procedure** RASTERIZE($H, G, vars, gridType$)
2: $(D, \Sigma) \leftarrow H$
3: $(\_, \Delta x, \Delta y, \Delta z) \leftarrow G$
4: $is3D \leftarrow \exists \sigma \in \Sigma \;.\; |V(\sigma)| = 4$
5: $n_{cells;x} \leftarrow \left\lceil \frac{|D_x|}{\Delta x} \right\rceil$
6: $n_{cells;y} \leftarrow \left\lceil \frac{|D_y|}{\Delta y} \right\rceil$
7: $grid_{out;vars} \leftarrow zeros\,(|vars|, n_{cels;x}, n_{cells;y})$
8: **if** $is3D$ **then**
9: $grid_{out;cnt} \leftarrow zeros\,(n_{cels;x}, n_{cells;y})$
10: **for** $\sigma \in \Sigma$ **do**
11: $rasterizeSimplex(\sigma, G, grid_{out;vars}, grid_{out;cnt})$
12: **end for**
13: **else**
14: **for** $\sigma \in \Sigma$ **do**
15: $rasterizeSimplex(\sigma, G, grid_{out;vars}, null)$
16: **end for**
17: **end if**
18: **return** $grid_{out;vars}$
19: **end procedure**

Each simplex σ in H, is then rasterized according to algorithm 2. Beginning in line 2, the axis-aligned bounding box (AABB) of the simplex is calculated using algorithm 3. An AABB of a body is the cartesian product of the intervals between the minimum and maximum coordinate of the body's points along each axis. Then, a sub-grid of G_{xy} is set up so we can count how many samples of σ in z-direction fall into each xy-cell. Additionally, the HNF, as discussed in section 2.1.2, is calculated for each face (edge) of the

Algorithm 2 The simplex rasterization algorithm

1: **procedure** RASTERIZESIMPLEX($\sigma, G, grid_{out;vars}, grid_{out;cnt}$)
2: $aabb \leftarrow boundingBox(\sigma)$
3: $aabbCells \leftarrow G_{xy} \cap aabb_{xy}$
4: $aabbCells_{cnt} \leftarrow zeros(|aabbCells_y|, |aabbCells_x|)$
5: $hnfs \leftarrow \{v \rightarrow HNF(V(\sigma) \setminus \{v\}, v)$ **for** $v \in V(\sigma)\}$
6: **for** $(x, y) \in aabbCells$ **do**
7: **if** $is3D$ **then**
8: **for** $z \in G_z \cap aabb_z$ **do**
9: **if** $pointInSimplex((x, y, z), hnfs)$ **then**
10: $aabbCells_{cnt}[y, x] += 1$
11: **end if**
12: **end for**
13: **else**
14: $aabbCells_{cnt}[y, x] = 1$
15: **end if**
16: **end for**
17: **for** $v \in vars$ **do**
18: **if** $gridType =$ **volume then**
19: $depthAveraging(grid_{out;vars}, grid_{out;cnt}, aabbCells_{cnt}, v)$
20: **else if** $gridType =$ **floor** $\wedge\ v = b$ **then** ▷ $b \triangleq bathymetry$
21: $depthRasterization(grid_{out;vars}, aabbCells_{cnt}, \sigma, v)$
22: **else**
23: $standardRasterization(grid_{out;vars}, aabbCells_{cnt}, v)$
24: **end if**
25: **end for**
26: **end procedure**

tetrahedron (triangle).

Now, following line 6, we test whether σ includes each 3D (2D) point in its AABB respectively. If a point is found to lie within the simplex, the sub-grid cell's counter at the point's (x, y) index is incremented (set to 1). After all points in the AABB have been processed, each variable in $vars$ is depth-averaged/depth-rasterized/standard-rasterized according to the criteria discussed in section 2.1.3 and the result is stored in $grid_{out;vars}$.

Note that in this algorithm, the values in the list $vars$ do not have an explicit data type and the only requirement for them is to provide a value for each simplex. This allows processing an *array of timesteps* at once for each variable as the geometry and topology of all grids are time-independent. This idea will be elaborated on in section 2.3.

The above loop over all simplices can furthermore be parallelized with $n_{threads}$ threads. Without any other modifications, the threads would, however, conflict with each other when writing their outputs. In section section 2.3.2, mechanisms will be put into place to avoid expensive locking and synchronization mechanisms. After each simplex has been rasterized, the final regular grid with rasterized variables is returned and can be written to the filesystem.

Algorithm 3 Bounding box (AABB) calculation

```
1: procedure BOUNDINGBOX(σ)
2:     vertices ← V(σ)
3:     return [min_x(vertices); max_x(vertices)]
4:                 ×[min_y(vertices); max_y(vertices)]
5:                 ×[min_z(vertices); max_z(vertices)]
6: end procedure
```

Furthermore, the point-in-simplex test can be implemented as shown in algorithm 4. Note that this test *fails early*, meaning that as soon as one face (edge) of the simplex is found of which the tested point is on the *outside*, the point is rejected. Since most points in the AABB will be outside the simplex, this can increase the algorithm's speed significantly.

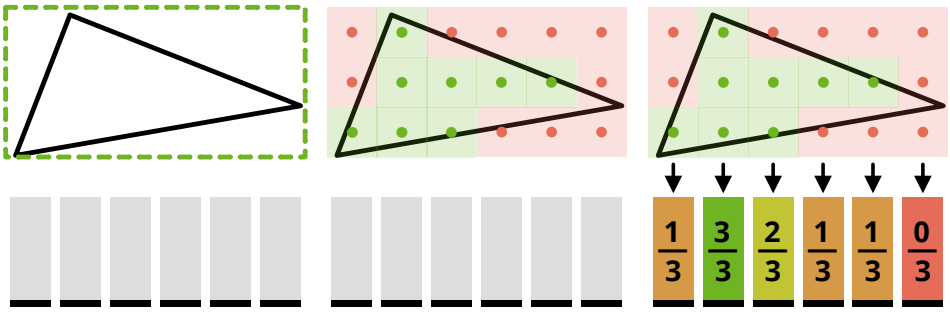
Algorithm 4 Point-in-simplex check

```
1: procedure POINTINSIMPLEX(q, σ)
2:     nVertices ← |V(σ)|
3:     for v ∈ V(σ) do
4:         (n, p) ← HNF(V(σ) \ v, v)
5:         if (⟨n, q⟩ + p < 0) ∨ (⟨n, q⟩ + p = 0 ∧ −n ≺ n) then
6:             return reject
7:         end if
8:     end for
9:     return accept
10: end procedure
```

Figure 2.6 shows the algorithm working on one tetrahedron.

2.3 Optimization for HPC Systems

In order to maximize the algorithm's performance on modern High Performance Computing (HPC) systems, a few considerations have to be made.



(a) *boundingBox* (b) *pointInSimplex* (c) *depthAveraging*

Figure 2.6: A simplex being rasterized onto a regular grid (bottom bars) according to the proposed algorithm. Planar view in y-direction, depth-averaging is shown. The rasterized variable in the simplex is considered to be 1 whereas the surroundings of the simplex are 0. The final shown values in (c) are the depth-average of the shown domain slice.

Some of them can also benefit consumer devices while others are required due to the high core count and distributed nature of HPC systems.

The CoolMUC2/3 clusters at LRZ feature a core count of 28-64 and RAM sizes between 64 and 96GiB per node and the algorithm is optimized for these architectures. Other specifications will be introduced as they become relevant.

2.3.1 Memory Management

Julia employs a Garbage Collector (GC) and dynamic allocations on the heap also decrease performance. Therefore it is beneficial to limit the amount of allocations and thus also GC work [2, ch. II.2].

We pre-allocate buffers/variables that would otherwise be re-instantiated over and over during rasterization and provide a return buffer to functions as an argument which is common practice in languages like *C*. Furthermore, the in-place variants of algorithms are used where possible for the same reason. Additionally, we can exploit the large RAM sizes mentioned earlier to work on multiple timesteps at once, without having to perform rasterization again for each one.

To establish an understanding of the algorithm's memory footprint, the sizes of all relevant data structures are listed in table 2.5.

This memory footprint is much larger than the amount of RAM available to the nodes of the previously mentioned HPC systems. To solve this issue, we can run the rasterization algorithm multiple times with only a subset

Table 2.5: The memory footprints of the data structures used by the rasterization algorithm. The example values have been calculated for a scenario with 7500×7500 cells in the regular grid, 4.500.000 tetrahedra and 1.000.000 points in the unstructured grid and 2 variables being rasterized across $n_{times} = 300$ timesteps.
These values are approximations of the ones obtained when rasterizing the ASCETE scenario with $\Delta x = \Delta y = \Delta z = 100\,\mathrm{m}$.

Data struct.	Footprint depending on input sizes	Ex. value
$vars$	$m_{vars;in} = sizeof(\texttt{Float64}) \cdot \lvert vars \rvert \cdot n_{times} \cdot \lvert H_\Sigma \rvert$	21 GiB
$geometry$	$m_{points} = sizeof(\texttt{Float64}) \cdot \lvert \{p \mid \exists \sigma \in H_\Sigma \,.\, p \in V(\sigma)\} \rvert \cdot 3$	23 MiB
$topology$	$m_{topo} = sizeof(\texttt{Int32}) \cdot \lvert H_\Sigma \rvert \cdot \lvert V(\sigma \in \Sigma) \rvert$	67 MiB
$grid_{out;cnt}$	$m_{cnt} = sizeof(\texttt{UInt16}) \cdot n_{cells;x} \cdot n_{cells;y}$	108 MiB
$grid_{out;vars}$	$m_{vars;out} = sizeof(\texttt{Float64}) \cdot n_{cells;x} \cdot n_{cells;y} \cdot \lvert vars \rvert \cdot n_{times}$	252 GiB
misc.	$m_{misc} = \cdots$	$4GiB$

of the timesteps being processed. These subsets should be kept as large as possible to reduce the number of iterations of the rasterization algorithm and therefore the number of repetitions of the same rasterization.

Due to the structure of the algorithm that requires an array for each simplex that contains its number of samples for each G_{xy}-cell in its AABB, it is not easily possible to cache the rasterization data efficiently. While this array is only instantiated once for each of the $n_{threads}$ threads and reused within them, the array would have to be stored for each simplex in order to to cache the rasterization which is not viable. Thus, the rasterization is performed multiple times, each time processing one or more timesteps.

We can calculate the maximum number of timesteps that can be processed per rasterization with a memory budget of m_{max} as follows:

$$n_{times;iter} = \left\lfloor \frac{m_{max} - m_{points} - m_{topo} - m_{cnt} - m_{misc}}{(m_{vars;in} + m_{vars;out}) \cdot n_{times}^{-1}} \right\rfloor , \tag{2.25}$$

which equates to $\lfloor 46.19 \rfloor = 46$ timesteps per iteration with the example values from table 2.5 and $m_{max;1} = 64GiB$. With $m_{max;2} = 96GiB$, the number of timesteps per iteration increases to $\lfloor 84.29 \rfloor = 84$.

Hence, the number of iterations needed is

$$n_{iterations} = \left\lceil \frac{n_{times}}{n_{times;iter}} \right\rceil , \tag{2.26}$$

which is 7 for $m_{max;1}$ and 4 for $m_{max;2}$.

2.3.2 Eliminating Resource Contention

Another optimization that can dramatically increase performance in multi-threaded applications is the minimization of contention, i.e. a thread having to wait for resources used by another thread [1, p. 295].

For this section, only write accesses both to RAM and to the filesystem will be considered as those are the only sources of contention in the discussed application.

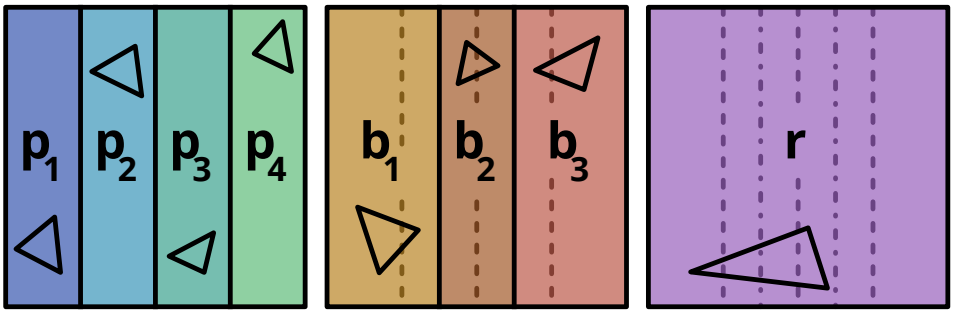
In the algorithm described in section 2.2.2, all input variables, i.e., the unstructured grid H and the input variables $vars$ are exclusively read from and never written to. The regular grids containing rasterized variables and the counter of samples per cell are, however, both read from and written to by all threads. This necessitates either a locking mechanism, or a method of ensuring that no two threads access the same area in those grids at the same time.

A locking mechanism introduces wait times for threads trying to acquire occupied locks and should therefore be avoided. It turns out that this is indeed possible:
If the simplices of the unstructured grid are not distributed across threads in an arbitrary fashion but are instead sorted into $n_{threads}$ partitions (also referred to *bins* or *buckets*) that do not overlap when projected onto the xy-plane, each thread can work on its own partition and will exclusively read from and write to its part of the output grids.

However, in a tetrahedral grid, it is not generally possible to define a rule to find such partitions because tetrahedra can overlap each other when projected onto xy.

To fix this issue, three separate partitionings of D_{xy} called $\mathcal{P}, \mathcal{B}$ and $\mathcal{R}$ are defined and processed in sequence in that order. In $\mathcal{P}$, D_{xy} is partitioned into axis-aligned rectangular bins which only include the simplices *fully inside* them (the left border of the rectangle being included, the right one being excluded).
The second partitioning $\mathcal{B}$ then divides D_{xy} such that the new partitions are centered on the edges of the previous ones and will therefore contain almost all of the simplices that were excluded before.
The residual simplices that spanned multiple partitions in both partitionings above are then processed sequentially in $\mathcal{R}$. Figure 2.7 shows these partitionings and example simplices in them.

We start out with $n_{threads}$ **p**artitions $p_1, \cdots, p_n$ of the domain D_{xy} which are each rectangular and span one or multiple complete columns of the output



(a) $\mathcal{P} = \{p_1, \cdots, p_n\}$ (b) $\mathcal{B} = \{b_1, \cdots, b_{n-1}\}$ (c) $\mathcal{R} = \{r\}$

Figure 2.7: The three partitionings of domain D_{xy} with an example simplex for each bin.

grid $G = (D, \Delta x, \Delta y, \Delta z)$. These partitions together cover all of D_{xy} and do not intersect each other (as required for a valid partitioning).

Next, an additional $n_{threads} - 1$ rectangular ***b**order* partitions $b_1, \cdots, b_{n-1}$ are defined such that each $b_i \in \{b_2, \cdots, b_{n-2}\}$ spans all columns between p_i's and p_{i+1}'s middle columns, the left middle column being included, the right one being excluded. b_0 spans all columns from the leftmost one to p_2's middle column (excluded) and b_{n-2} spans all columns from p_{n-1}'s middle column (included) to the rightmost one.

A final partition for ***r**esidual* simplices covering all of D_{xy}, r, is then defined.

With these partitionings, all simplices end up in at least one partition: Partition r contains *all* simplices in D that are not included in any of the bins in $\mathcal{P}$ and $\mathcal{B}$.

2.3.3 Load Balancing

Maximum throughput is only achieved when all threads take the same time when processing the partitions in $\mathcal{P}$ or $\mathcal{B}$ respectively such that idle time is minimized. Additionally, the r-bin should be (close to) empty as it is processed by only one thread while the others are idle.

Since the bins in each partitioning have to be non-overlapping and the partitionings need to be processed after another, it is much easier to statically balance the loads before rasterization as opposed to a dynamic approach.

The time each thread needs does not translate directly to the number of simplices that thread has to process:
Since simplices have different sizes and proportions, the number of regular grid cell centers that have to be tested with the $pointInSimplex$-function,

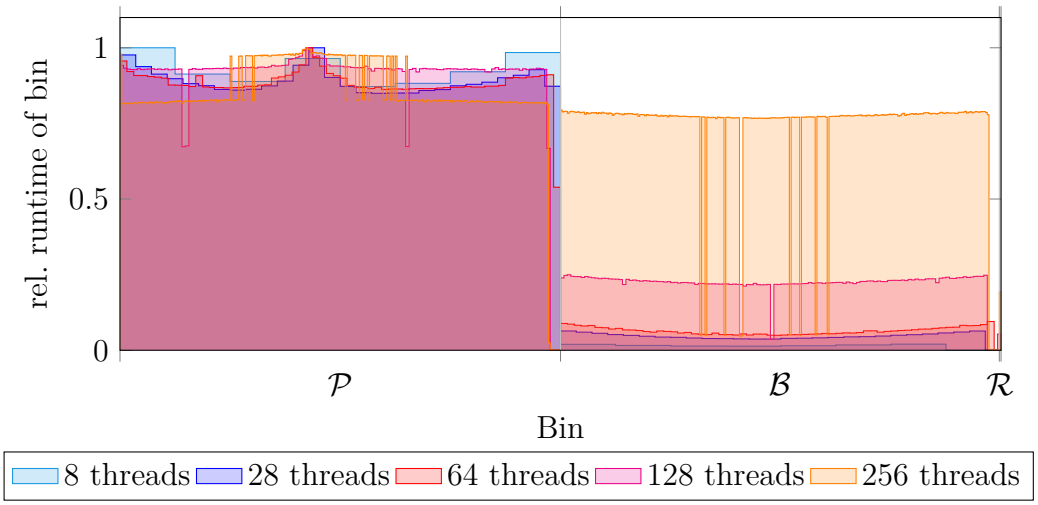


Figure 2.8: The runtime of each bin (each bin is one column), relative to the longest bin runtime observed, for different $n_{threads}$. For each number of threads, the bin sizes are different and this is reflected here: the bins for $n_{threads} = 8$ are visible individually while those for $n_{threads} = 256$ are barely distinguishable. These are the distributions as obtained by binning the **volume** grid of the ASCETE scenario. The b-bins are appended on the right of the p-bins and the r-bin is the last value in each plot.

and therefore the computational demand for each simplex, varies.

Figure 2.8 shows that the naive approach of sizing each p-bin equally like shown in fig. 2.7a already produces reasonably balanced runtimes and thus no further optimizations are made here.

Another idea for load balancing would be to estimate the actual computational load (read: runtime) for the different simplex sizes in the grid and then resize each bin to even out the load even further. This has proven to be very difficult to do well and the results measured there have not surpassed those of the naive approach. Therefore these results are not discussed further here.

2.3.4 Eliminating Random File System Accesses

The last important optimization discussed in this chapter is the elimination of random file system accesses. The parallel file system used for large in- and output data at the LRZ has much higher latencies compared to a typical SSD as found in modern PCs but similarly high bandwidth (when using a single compute node); This is visualized in fig. 2.9.

While the difference between random and sequential accesses on a modern

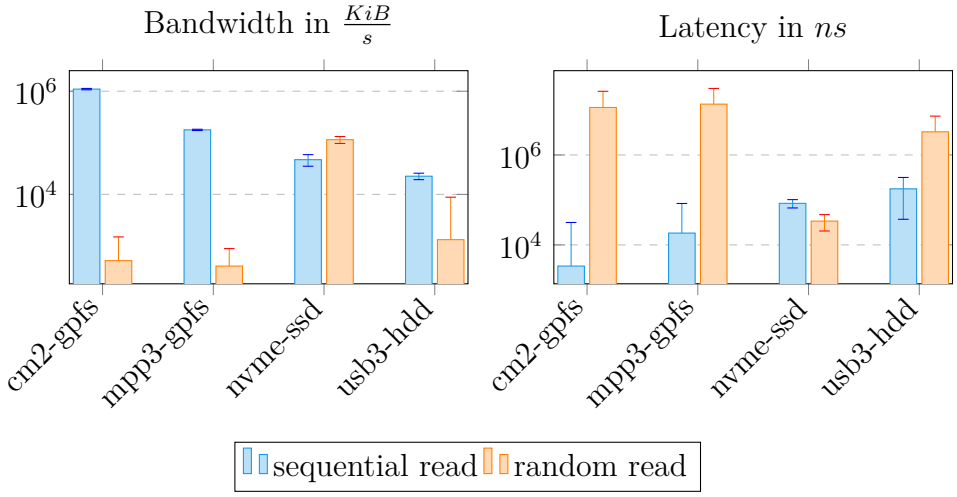


Figure 2.9: Comparison of sequential and random read performance of different storage systems. Error bars show standard deviation over three separate runs with $2min$ of runtime each. All performance testing has been done with *fio* (Flexible IO-Tester, ver. 3.20, `https://github.com/axboe/fio`). "cm2-gpfs" and "cm3-gpfs" refer to the GPFS (IBM Spectrum Scale) filesystem accessed from CoolMUC2/3 respectively.

SSD is almost negligible, accessing storage randomly instead of sequentially can lead to unacceptably slow performance on the targeted HPC systems.

We already rasterize in simplex-order (see section 2.2.1) for this exact reason and we can also write the output arrays $grid_{out;vars}$ sequentially. The only relevant file system accesses left that still need to be optimized are the ones to $vars$ during rasterization. Since this is done in a multi-threaded environment these accesses are not sequential despite simplex-order rasterization: Each thread has its own set of simplices within which it operates and because of the binning step there is no guarantee that simplices within that set are a continuous sequence. Additionally, as discussed in section 2.3.3, the procesing speed varies between simplices and each thread can progress through its bin at a different rate.

To resolve this issue, instead of accessing the variables in $vars$ through the file system, we load them into a buffer *sequentially* before the next rasterization iteration begins. This data can then be accessed *randomly* from RAM with a much smaller performance penalty.

The memory footprint of the $vars$-buffer is hinted at in table 2.5 but of course has to be divided by the number of rasterization iterations performed.

2.4 Evaluation

With all of the abovementioned optimizations, the rasterization algorithm takes $\approx 21\,\text{min}$ to rasterize all 300 timesteps of the ASCETE scenario into a regular grid of 7500×7500 cells on a single CoolMUC-2 node. Please note that at the time of writing, the program is not able to utilize the full RAM of the node due to garbage collection issues. Thus, a memory limit of $25GiB$ has been imposed and the program needs six iterations for each of the triangle grids and 12 iterations for the tetrahedral grid (see section 2.3.1).

As discussed in section 2.1.4, the relative error between the in- and output grids, averaged over the whole domain, is $\approx 1.3\,\%$ when rasterizing tetrahedra and $\approx 0.17\,\%$ when rasterizing triangles. This error might be higher in timesteps where there are many discontinuities near the fault. Thich is generally not a problem since the later timesteps (those with the maximum seafloor displacement or those immediately after the earthquake) are those that are the starting point for tsunami simulations.

As stated in section 2.2, the program can benefit from distributed parallelization (e.g. by having each cluster node work on a subset of the timesteps) but this functionality has not been implemented here. The runtime of 21 minutes is fast enough for most use cases and bigger scenarios should be processed fast enough, too.

Chapter 3

The Effect of Initial Velocity on Tsunami Simulations

The Shallow Water Equations usually require a state vector $\mathbf{Q} = (b, h, \mathbf{p})^T$ for each cell of the domain to store the simulation's state from which the next timestep can be calculated. b is the bathymetry height, h the height of the water column and $\mathbf{p} = h\mathbf{v}$ the two-dimensional momentum vector. To initialize the simulation these values have to be set for each cell. It is common practice to only set b and h to the actual initial values and to set $\mathbf{v}$ to $\mathbf{0}$.

The approach in the previous chapter allows researchers to either only rasterize the time-dependent vertical seafloor and sea-surface displacements in each raster cell or to additionally rasterize the depth-averaged water velocity in the x- and y-directions.

This leads to the question whether or not this additional data improves tsunami simulation results and therefore if it is advisable or even necessary to include it as an initial condition.

3.1 Related Work

The debate of whether initial velocities are negligible or not has been long-lasting and examples for each standpoint have been brought forth:
While [23, 21, 27, 22] come to the conclusion that initial velocities have an important effect in tsunami genesis by comparing simulation results to real-world tsunami data, [12] concludes that the effect of initial velocity is negligible.

The case for initial velocities

In [23, 21], Song investigates the 2004 Sumatra-Andaman earthquake and takes satellite data, such as GPS readings and tsunami heights as well as seismographic data into account, coming to the conclusion that the initial momentum introduced by the sloping seafloor has a significant effect on the resulting tsunami height and energy.

They go on to develop an early inundation prediction method also using GPS and the DART[1] (Deep-ocean Assessment and Reporting of Tsunamis) network of sensors and validate it using data from the Tohoku 2011 tsunami. Once again they state the importance of initial velocity for tsunamis, claiming that initial velocity caused about half of the vertical sea surface displacement at certain points of measurement. [27, 25]

In [22] they revisit the 2011 and 2004 tsunamis mentioned above and compare their theory of initial momentum (kinetic energy) making up a significant portion of tsunamis' total energy to measured real-world and lab experiment data. Once again, they find that the consideration of initial momentum is important for the mentioned tsunamis.

Both of these tsunamis feature a small amount of vertical uplift and a large amount of horizontal displacement which is assumed to be the cause of the high initial momentum.

The case against initial velocities

Lotto et al. [12] compare a fully-coupled simulation of an earthquake and the resulting tsunami with an ordinary shallow-water tsunami simulation with varying amounts of initial velocity. They show that as this velocity approaches 0, the results of the SWE simulation have the smallest misfit compared to the fully-coupled simulation.

3.2 Results

When including velocity data as tsunami simulation inputs, we cannot anymore utilize time-dependent displacements but rather have to choose *one timestep* after the earthquake rupture process is complete as the initial condition. This is because SWE solvers update water impulse and height in each cell and thus we would either have to overwrite the impulse with the input data or develop a technique for merging the simulation and input data. This is not a problem that occurs for time-dependent seafloor displacements

[1] `https://nctr.pmel.noaa.gov/Dart/`

since the tsunami simulation is only coupled in one direction: the seafloor displacements have an effect on the water body but the water has no effect on the seafloor.

In order to perform the following tsunami simulations, the version of sam(oa)^2 with initial velocity support is needed. Refer to appendix A.2 for further information.

To evaluate the effect of initial velocity on the resulting simulated tsunami, simulation runs of the ASCETE scenario with different initial values will be compared:

- (d, η, v) seafloor displacement, surface displacement, and initial velocity at $t = 300\,\mathrm{s}$,
- (d, η) seafloor and surface displacement at $t = 300\,\mathrm{s}$,
- (d, v) seafloor displacement (copied to surface) and initial velocity at $t = 300\,\mathrm{s}$,
- (d) seafloor displacement (copied to surface) at $t = 300\,\mathrm{s}$,
- $(d\ (timedep))$ time-dependent seafloor displacements for $0\,\mathrm{s} \leq t \leq 300\,\mathrm{s}$ (copied to surface).

Note that in real-world scenarios, some of these data points might not be available and thus compromises have to be made, e.g. copying the seafloor displacements to the surface (see chapter 4).

In theory, the first run in the list is an exact snapshot of the fully-coupled earthquake-tsunami simulation at $t = 300\,\mathrm{s}$ and is therefore used as the baseline for comparison.

Figure 3.1 shows the sea surface elevation along the x- and y-axis which intersect at the epicenter, at the end of the abovementioned simulation runs. Note, however, that the start time of some runs had to be shifted in order to account for the faster propagation of waves *with* initial velocity compared to those which first need to build momentum through gravitation. This has been done by aligning the leading wave peaks from all runs for the latest timestep available; The shifts are noted in the figure.

The result of this comparison is that it does not seem to matter much if a tsunami simulation is run with (d, η, v), (d) or $(d\ (timedep))$ as initial conditions. The only combinations that deviate from those three wave profiles are the ones where either initial velocity was used together with the bottom deformation copied to the surface or where the seafloor and sea surface displacements from SeisSol were used but initial velocity was discarded.

In the case of the (d, v)-scenario, the impulse is much higher than in

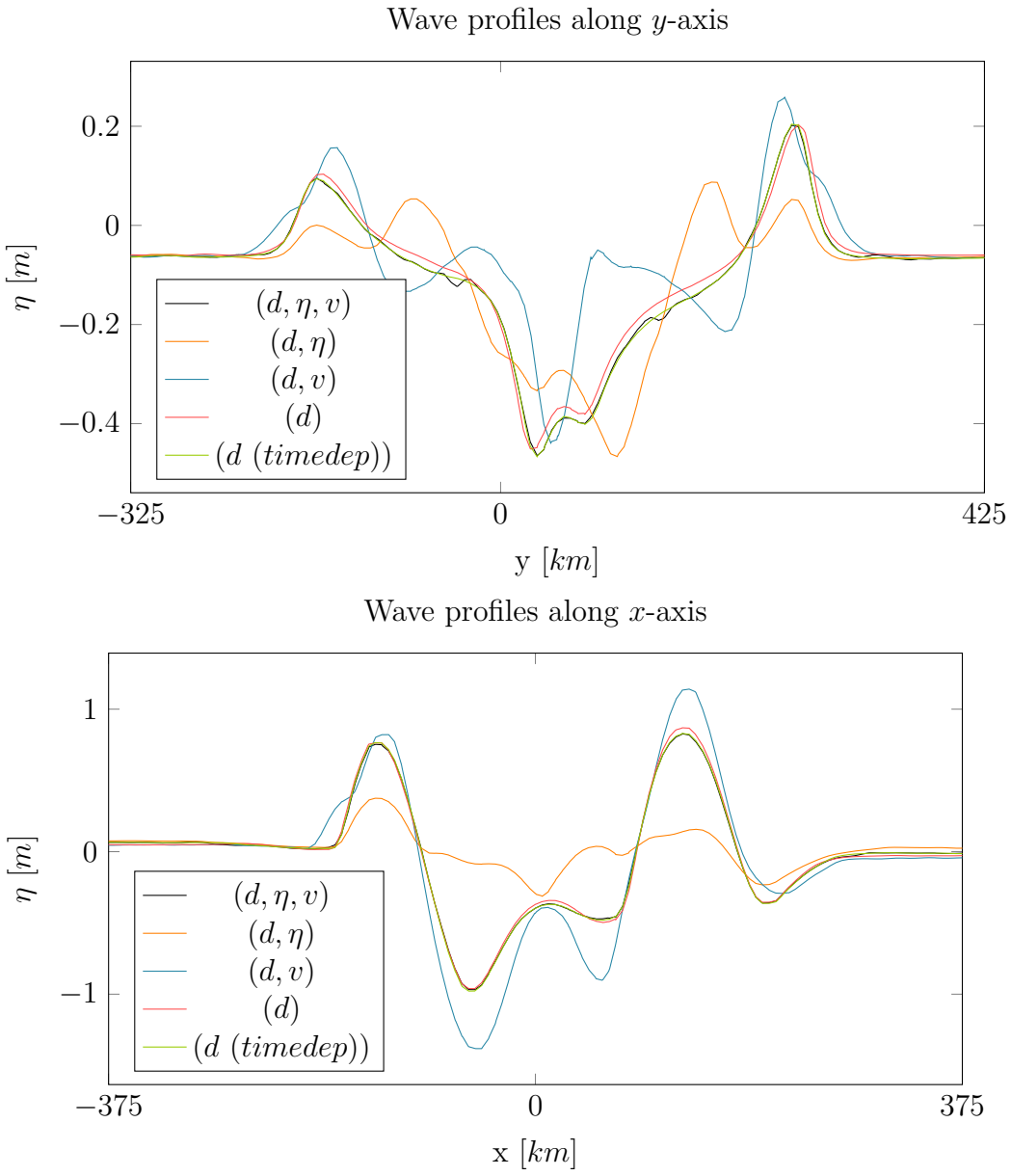


Figure 3.1: Comparison of wave profiles along major axes at $t = 868\,\mathrm{s}$ (the last timestep available after shifting start times of some runs). (d, η, v), (d) and $(d\ (timedep))$ agree well with each other while the waves in (d, v) are of much bigger amplitude and those in (d, η) are of much lower amplitude and significantly other shape. Start times were adjusted to synchronize wave profiles at the mentioned timestep: (d, v): 160 s earlier, (d): 32 s later. The temporal resolution of the outputs is $\sim 4\,\mathrm{s}$ which does not allow for *perfect* synchronization of these runs. However, this is still enough for qualitative and rough quantitative comparisons.

the other scenarios since the water height is a constant $-b = 2000\,\mathrm{m}$ (the displacements are copied to the surface, thus conserving the water height from before the displacement) but the velocity is taken from timestep $t = 300\,\mathrm{s}$ and is thus already quite high. This results in a much too high initial impulse and therefore this combination of initial conditions is invalid.

As for the (d, η)-scenario, the opposite is the case: The water height is taken from $t = 300\,\mathrm{s}$, i.e. the initial potential energy had 300 s to partly be turned into kinetic energy, and this kinetic energy is then reset to zero. This results in a much too shallow wave profile and is also an invalid combination of initial conditions.

The $(d\ (timedep))$-scenario is slightly closer to the (d, η, v)-scenario in the later timesteps than the (d)-scenario although this might differ from earthquake to earthquake: The peaks of the wave profiles shown in fig. 3.1 are up to $\sim 5\,\mathrm{cm}$ higher in the (d)-run than in the (d, η, v)-run while those of $(d\ (timedep))$ are within 1 cm of (d, η, v). The $(d\ (timedep))$-approach is, in fact, a slightly more realistic approach than only using (d) as the initial condition since the former already simulates the tsunami *while* the earthquake is still in progress. The latter approach represents the earthquake as an instantaneous event which is, of course, not in line with reality.

Tsunami simulations with initial conditions taken from earlier timesteps have been performed as well: [13] states that the ASCETE scenario A reaches almost constant seafloor displacements at $t = 80\,\mathrm{s}$ with only surface waves (on the seafloor's surface, not the sea surface) still travelling through the domain. However, the displacements from those surface waves cause significant artifacts in the wave profile which are not present with either of the initial conditions described above.

The verdict whether to use initial velocities or not is: *If* seafloor *and* sea surface displacements are known for a timestep *shortly after* surface waves have left the perimeter of the earthquake initial velocities can improve the tsunami simulation slightly. It is possible that *some* tsunamigenic events such as those with moving slopes mentioned by Song et al. [23, 21, 27, 25, 22] are affected significantly by the inclusion or exclusion of initial velocities and it is also possible that non-SWE tsunami solvers might be affected. These investigations were, however, not in the scope of this thesis.

If sea surface displacements are unknown, measured initial velocities are of no use since they cause errors in the initial momentum and thus in the later wave profile.

Time-dependent seafloor displacements provide a very good approximation to simulations where all initial conditions are known.

Chapter 4

Depth-filtering

4.1 Motivation

Many tsunami simulations use seafloor displacements copied 1:1 to the water surface as an initial surface displacement. This is, of course, inaccurate as the water is not displaced strictly in an upwards direction but also horizontally [10].

The direct copying of those displacements leads to high-frequency components being present in the surface elevation which *can* lead to large errors later in the tsunami simulation due to frequency dispersion [8, 16].

It is worth noting that full 3D approaches such as the simulation of the water body in SeisSol solve this problem by simulating the acoustic wave equation for each volume element in the 3D water body [11] and therefore simulating the effects the seafloor displacement has on the sea surface accurately.

However, when such simulations cannot be employed or if only part of their output data is available, one might have to resort to a method of approximating the real sea surface deformation from the seafloor displacements given.

This is where *depth filtering* is used, as first described by [10] for a flat and infinite seafloor geometry with a method called *Kajiura's Filter* in this thesis. Depth filtering takes the (vertical) seafloor deformation D_{ij} at any given point as well as the water height H_{ij} and the dispersion time [8] τ as inputs and outputs the surface displacements η_{ij} – no water velocities are needed.

4.2 Related Work

Kajiura 1963

The original paper [10] describes the mathematical principles behind depth filtering and shows the restrictions of Kajiura's Filter. Kajiura provides an approach for depth filtering during tsunami genesis and propagation, although we are only interested in the genesis phase. The proposed methods rely on several assumptions:

1. constant seafloor depth,
2. no lateral boundary ($\hat{=}$ infinite ocean),
3. instantaneous bottom deformation.

While not all of these (and in fact, *none* of these) can be applied to real-world scenarios, there are still scenarios where these assumptions approximately hold.

Kajiura's Filter is discussed in detail in section 4.3.1.

Saito & Furuma 2009

Saito and Furuma investigate different methods of simulating tsunami generation including a full 3D simulation of the Navier-Stokes-Equations and Kajiura's Filter [19]. They provide a table which shows the applicability and necessity of different transformations of the sea surface displacement for different parameters of the scenario.

They also describe how non-instantaneous bottom displacements can be used with Kajiura's Filter.

Glimsdal et al. 2013

The authors of [8] investigate whether the effect of *frequency dispersion* - i.e. waves of different wavelengths transporting energy at different speeds [8, p. 1508ff] - is important or negligible for tsunamis.

To do so they employ models for tsunami generation by earthquakes and landslides and the earthquake model includes depth filtering to get rid of unnatural high-frequency components in the surface displacements [8, sec. 3.1].

The used depth filter is Kajiura's Filter and a detailed description of how to implement it numerically is given. Also, a short comparison with other methods of obtaining the surface displacements is made.

They conclude that tsunamis caused by "moderate-magnitude earthquakes" [8, p. 1523], landslides and events with high dispersion times τ have significant dispersion effects while ones generated by earthquakes of high magnitude do not.

This means that for the former tsunamis, the effect of erroneously included high-frequency components in the surface displacement will have a higher effect on the inundation results than for the latter.

Margottini et al. 2013

In [14], Margottini et al. investigate the 1888 Trondheim tsunami which was caused by a submarine landslide. They employ Kajiura's filter to obtain more realistic wave patterns than otherwise possible and then compare different solvers regarding their ability to handle dispersive effects.

They claim that the simulations with depth-filtered displacements are more in line with observations of the tsunami compared to previous simulations.

4.3 Kajiura's Filter

4.3.1 Theoretical Explanation

This chapter will *not* provide the physical explanations needed to fully understand depth filtering as this is both outside the scope of this thesis and explained very thoroughly by Kajiura himself in [10].

Instead, the mathematics necessary to *perform* depth-filtering are discussed here.

The Filter

According to [10, p. 541ff], the surface displacement of the water body is determined by the time and by multiple initial conditions at both the seafloor and the sea surface:

1. water velocity at surface,
2. initial elevation of surface,
3. surface pressure,
4. elevation magnitude at the bottom,
5. elevation speed at the bottom.

We are interested in the special case (c) in [10, p. 541] where an instantaneous bottom deformation happens in an otherwise resting water body. This means that only the elevation magnitude at the bottom remains as an initial condition.

Kajiura then provides an explicit formula for the initial surface displacement for the abovementioned parameters:

$$R(\bar{r}) = \frac{1}{\pi} \sum_{n=0}^{\infty} \frac{(-1)^n (2n+1)}{((2n+1)^2 + \bar{r}^2)^{\frac{3}{2}}} \tag{4.1}$$

where R is the vertical surface displacement at a point (x, y) created by a vertical displacement of the seafloor at point (x_0, y_0) with $\bar{r}^2 = (x - x_0)^2 + (y - y_0)^2$.

He also shows that the area of influence of a point displacement at the bottom has a radius relative to the water height H which leads to Margottini et al. ignoring the influence of bottom displacements on points that are farther than $5H$ from the source [14, p. 75].

To get the total surface displacement at (x, y), one simply has to sum up the contributions R from every bottom displacement in the area of influence.

As we are not working in a continuous domain but rather on a regular grid, we have to do further steps until we can use this approach. These are described in detail in [14, p. 75] and [8, p. 1511] and lead to the formula

$$\eta_{ij;kl} = \sigma \frac{\Delta x \Delta y}{H^2} D_{ij} R(\frac{\bar{r}^2}{H^2}) \tag{4.2}$$

with $\bar{r} = (x_i - x_k)^2 + (y_j - y_l)^2$ being the distance between two grid points with indices (i, j) and (k, l) where (i, j) is the point of the bottom deformation and (k, l) is the point of the resulting surface displacement.

D_{ij} denotes the magnitude of the bottom displacement and σ is chosen such that the displaced volume at the surface equals the displaced volume at the bottom.

The final formula for the total surface displacement at a point (k, l) given all bottom displacements in the area of influence is then (adapted from [8, p. 1511] to match style with the above formulae):

$$\eta_{kl} = \sum_i \sum_j \frac{\Delta x \Delta y}{h_{ij}^2} \sigma_{ij} D_{ij} R(\frac{\bar{r}^2}{h_{ij}^2}) \,. \tag{4.3}$$

Note that Glimsdal et al. permit a variable seafloor height h_{ij} which is theoretically incorrect as they state themselves, but they also state that eq. (4.3)

approximates the correct solution well if the bathymetry has little variation over the distance of multiple ocean depths [8, p. 1511].

To obtain $\sigma_{i_0 j_0}$, we first only consider a scenario with *one* cell with nonzero bottom displacement $D_{i_0 j_0}$ and the cells in its area of influence along with their resulting surface elevation.

When we build the sum over all surface displacement volumes $\Delta x \Delta y \eta_{kl}$ in this area, we get the total volume displaced at the surface. It is clear that in the case of only *one* $D_{i_0 j_0} \neq 0$, all of the displacement of the surface has to come from this one $D_{i_0 j_0}$. Since we preserve volume (Kajiura assumes an incompressible ocean [10]), these two volumes have to be equal, or put differently:

$$\begin{aligned} & \Delta x \Delta y \sum_k \sum_l \eta_{kl} = \Delta x \Delta y D_{i_0 j_0} \\ \iff \quad & \sum_k \sum_l \eta_{kl} = D_{i_0 j_0} \,. \end{aligned} \tag{4.4}$$

From eq. (4.3) and eq. (4.4) we can now obtain $\sigma_{i_0 j_0}$:

$$\begin{aligned} \eta_{kl} &= \sum_i \sum_j \frac{\Delta x \Delta y}{h_{ij}^2} \sigma_{ij} D_{ij} R(\frac{\bar{r}^2}{h_{ij}^2}) \\ &= \frac{\Delta x \Delta y}{h_{i_0 j_0}^2} \sigma_{i_0 j_0} D_{i_0 j_0} R(\frac{\bar{r}^2}{h_{i_0 j_0}^2}) \\ \Rightarrow D_{i_0 j_0} &= \sum_k \sum_l \frac{\Delta x \Delta y}{h_{i_0 j_0}^2} \sigma_{i_0 j_0} D_{i_0 j_0} R(\frac{\bar{r}^2}{h_{i_0 j_0}^2}) \\ \iff \quad \sigma_{i_0 j_0} &= \frac{h_{i_0 j_0}^2}{\Delta x \Delta y \sum_k \sum_l R(\frac{\bar{r}^2}{h_{i_0 j_0}^2})} \,, \end{aligned} \tag{4.5}$$

which is constant for constant height and can be tabularized for variable height.

Figure 4.1 shows the Kajiura-filter's response at the sea surface for a rectangular bottom deformation on a flat seafloor.

4.3.2 Applicability to Real-world Scenarios

As explained before, Kajiura's Filter assumes a perfectly flat and infinitely wide seafloor – which is non-existent in the real world. However, Saito and Furuma [19] provide a table with constraints for a scenario such that Kajiura's filter can be applied with reasonable accuracy. Furthermore, Kajiura expects

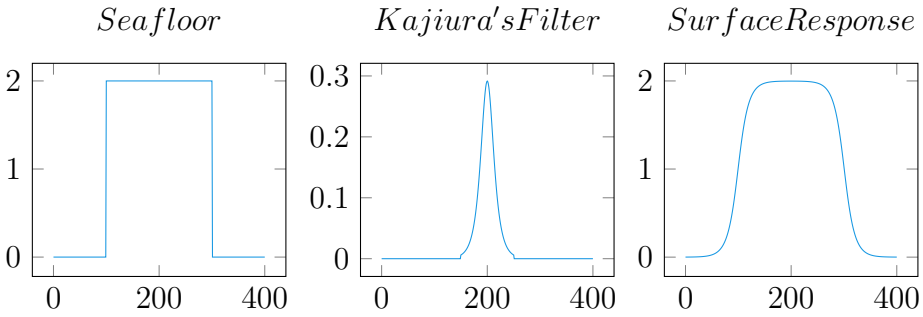


Figure 4.1: Kajiura's Filter applied on a rectangular displacement of the flat seafloor ($h = 2000m, \Delta x = 100m$). Horizontal axis shows cell index, vertical axis shows vertical displacement in meters (except for filter kernel). Left: The bottom displacement ($d = 2m$ in the center $\frac{10H}{\Delta x} + 1$ cells, $0m$ otherwise). Middle: The filter kernel (scaled up in y-direction, unit-less). Right: Surface response.

the deformation at the seafloor to happen *instantaneously* which is impossible in the real world. Saito and Margottini [19, 14] therefore propose applying Kajiura's Filter on the difference in bottom elevation for each timestep of the earthquake and adding the filtered contribution to the sea surface in each of those timesteps while running the tsunami simulation, which can be interpreted as a series of instantaneous (independent) displacements.

This requires the extension of the used tsunami simulator to allow for said surface displacement being different from the seafloor displacement. Since Kajiura's Filter ensures (via σ_{ij}) that the displaced volume at the surface equals the one on the seafloor, we can adjust the water height in each timestep accordingly:

$$h_{ij;t+1} = h_{ij;t} - \underbrace{(b_{ij;t+1} - b_{ij;t})}_{\Delta b} + \underbrace{(\eta_{ij;t+1} - \eta_{ij;t})}_{\Delta \eta}, \tag{4.6}$$

with $b_{ij;t}$ being the bathymetry (*including seafloor displacement*) at timestep t and $\eta_{ij;t}$ being the surface displacement at that timestep. The incremental approach with Kajiura's Filter delivers those differences Δb and $\Delta \eta$ directly for each timestep.

4.3.3 Numerical Implementation

The function R from eq. (4.1) is, unfortunately, an infinite sum. However, we can approximate the value of R with a sum over $0 \leq n \leq n_{iter;R}$ instead since

the sum's denominator grows quicker than its numerator. We choose $n_{iter;R}$ such that it is large enough to reduce the error of this simplification to an acceptable level and small enough to keep computation time at a reasonable level. The latter constraint is not a problem since we can tabularize R, which only depends on $\bar{r}$ which in turn depends on water height and the distance of "source" and "destination" grid point. The evaluation section 4.4 below gives actual values for $n_{iter;R}$ and the resulting filter error.

As stated before, we only need to consider the effect of a bottom displacement for grid points up to $n_H \cdot H$ from its position. Margottini et al. [14] use $n_H = 5$ which will be shown to introduce a relatively large error in section 4.4.

The table needs to contain values for R in the $[0; \frac{n_H \cdot H}{H}] = [0; n_H]$ range (or up to $\sqrt{n_H^2 + n_H^2}$ when working with a square-shaped area of influence, which we opt to do). R can then be interpolated between the table's values to make the function continuous and to reduce the needed number of table entries.

Similarly, σ can be tabularized according to eq. (4.5) by its height, as mentioned before, and interpolation can be applied as well. In scenarios of constant height, only a single computation of σ is needed.

With these pre-computed values, we can now apply eq. (4.3) to each grid point. If the scenario has multiple timesteps we have to apply the filter to the *difference* in bottom displacement each timestep has to its predecessor and save the result for each timestep. This yields the *difference* in surface displacement which can then be utilized by the tsunami simulator according to eq. (4.6).

4.3.4 Optimization

FFT-based implementation

The "naive" implementation of Kajiura's Filter is unbearably slow because of its $\mathcal{O}(n_x^2 \cdot n_y^2 \cdot h^2)$ runtime. A better approach is to model the filter as a convolution of a 2D grid of displacements F and the 2D filter G and to then make use of the convolution theorem [4, p. 70] to achieve a $\mathcal{O}(n_x \log n_x \cdot n_y \log n_y)$ runtime.

To do this, we first dissect eq. (4.2):
The part $f_{ij} := \sigma \frac{\Delta x \Delta y}{H^2} D_{ij}$ is evaluated for each cell of the domain and forms the matrix F:

$$F := \begin{pmatrix} f_{1,1} & \dots & f_{1,n_y} \\ \vdots & \ddots & \vdots \\ f_{n_x,1} & \dots & f_{n_x,n_y} \end{pmatrix}$$

while $R\left(\frac{\bar{r}^2}{H^2}\right)$ is evaluated in the $n_H H$-radius discussed above:

$$G := \begin{pmatrix} R(\sqrt{2} \cdot n_H) & \dots & R(n_H) & \dots & R(\sqrt{2} \cdot n_H) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ R(n_H) & \dots & R(0) & \dots & R(n_H) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ R(\sqrt{2} \cdot n_H) & \dots & R(n_H) & \dots & R(\sqrt{2} \cdot n_H) \end{pmatrix}$$

which is the filter kernel of the convolution. Care has to be taken to add zero-padding around both matrices to bring both of them to a size of $(n_x + \sqrt{2} \cdot n_H) \times (n_y + \sqrt{2} \cdot n_H)$ to avoid the aliasing introduced by the convolution in frequency space [3, p. 211]. We can then calculate the resulting surface displacement grid: $H = F * G$, with $*$ being the discrete 2D convolution operator.

This, however, requires us to sacrifice the support for variable water height since the filter G has be constant and its value and size depend on the water height: $|G| = \left\lceil \frac{2 \cdot n_H H}{\Delta x} + 1 \right\rceil \cdot \left\lceil \frac{2 \cdot n_H H}{\Delta y} + 1 \right\rceil$.

According to the convolution theorem, we can calculate H using the (Inverse) Fast Fourier Transform ((I)FFT) and a multiplication of the two Fourier-transformed matrices: $H = IFFT(FFT(F) \cdot FFT(G))$.

Parameter selection

To establish a valid baseline for the evaluation of the filter on actual earthquake scenarios we will first start with finding the best parameters n_H and $n_{iter;R}$ for the filter. Figure 4.2 shows possible parameter values and their relative error in volume compared to the displacement at the seafloor.

Since the runtime of the filter does *not* depend on h or n_H anymore with the FFT-based implementation, there is no (multiplicative) cost for using large values of n_H. Similarly, R is only tabularized once (or could even be hardcoded) and thus high $n_{iter;R}$ values do not cause a significant increase in runtime either. Knowing this, $n_H = 20$ and $n_{iter;R} = 10^4$ will be used

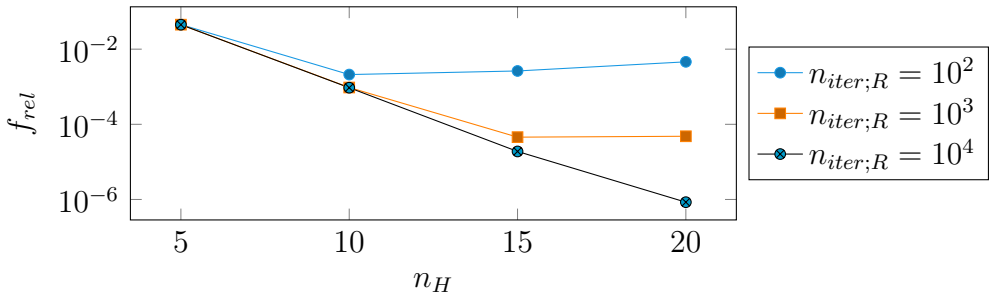


Figure 4.2: The relative errors of the displaced volume at the surface relative to the seafloor displacement for different parameters.

for the rest of the thesis. Note that for $n_H = 5$ the relative error is always approximately 4.5% which is *sizeable*. However, this is a necessary sacrifice to make when not using the FFT-based implementation (e.g. for non-constant bathymetry) since n_H acts as a multiplicative cost factor in that case.

4.4 Evaluation

The tsunami simulations performed in this section require a modified version of sam(oa)^2 that supports initial velocities and sea surface displacements. Refer to appendix A.2 for more information.

Now that we have minimized the volume error Kajiura's Filter introduces, we can test the filter on the ASCETE scenario:
Figure 4.3 shows that the difference Kajiura's Filter makes in the ASCETE scenario is minor. Taking the lower magnifying glass in the y-axis plot as an example, the height difference between $(kaj\ (timedep))$ and (d, η, v) is 5 mm while $(d\ (timedep))$ differs by 11 mm from (d, η, v). This is the most pronounced difference in the whole timestep and Kajiura's Filter still leads to a slightly closer approximation than time-dependent displacements over most of the shown wave profiles.

Since sam(oa)^2 and SWE solvers in general do not simulate frequency dispersion [8, p. 1509], the observed difference might be smaller than it would be in a dispersive solver [19].

Furthermore, Kajiura's Filter is not applicable to some scenarios where a tsunami is generated and thus care needs to be taken to validate its usage for a given scenario. Saito [19] provides conditions for the valid usage of Kajiura's Filter.

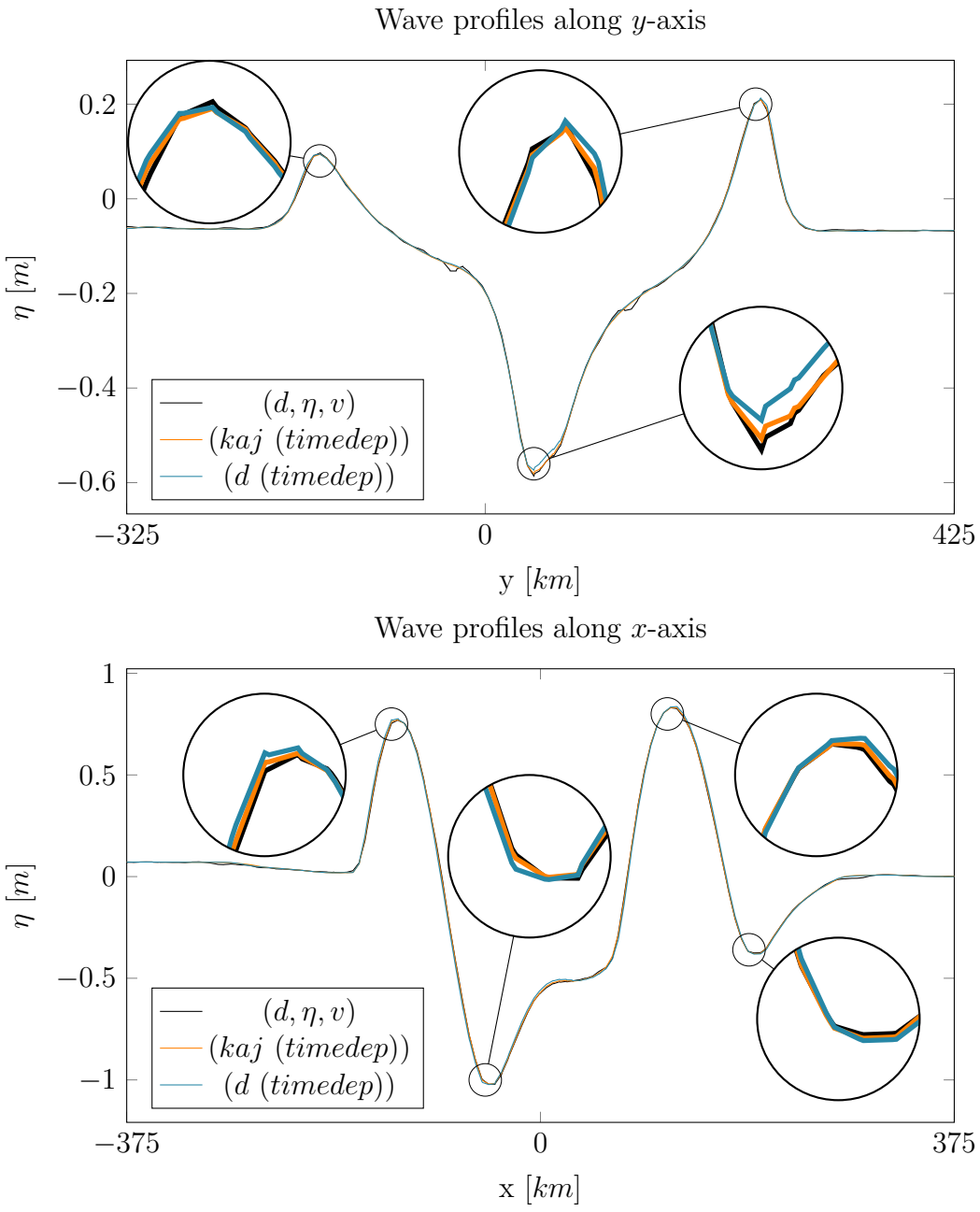


Figure 4.3: Comparison of wave profiles along major axes at $t = 868\,\mathrm{s}$ (in order to be comparable with fig. 3.1). All of the wave profiles lie very close together with Kajiura's Filter resulting in a slightly closer approximation to (d, η, v) than $(d\ (timedep))$ is.

Chapter 5

Conclusion

The rasterization of simplex meshes is a problem mostly solved for highly parallel architectures (GPUs). Rasterizing on the CPU is necessary for many HPC systems which do not employ GPUs and it turns out that the process is fast once file system accesses and memory management have been optimized.

The simplification of only sampling simplices at regular grid cell centers introduces a rasterization error, which is about 1.3% for tetrahedra when averaged over the domain of the tested scenario and 0.17% for triangles.

This error is reduced by using higher-quality meshes and with the unstructured grid's variables being smooth functions. Even when simplices are so small that they are not rasterized, their surroundings of similar value will be and therefore the error is kept small there, too.

Rasterization can be sped up significantly by binning simplices into disjunctive buckets such that no thread synchronization is needed and by utilizing the full amount of memory available by processing multiple timesteps at once.

Initial velocities can lead to a more accurate tsunami simulation. However, the sea surface displacements and seafloor displacements both have to be known at the same timestep of these velocities since the initial momentum of the tsunami gets distorted otherwise. Also, knowing seafloor and sea surface displacements without the velocity at that point also leads to wrong initial momentum and thus yields a wildly different simulation result.

Copying only the final seafloor displacements of the tsunamigenic event to the surface results in an almost identical wave profile across timesteps but a time delay is introduced when compared to a simulation with initial velocities. Using time-dependent seafloor displacements is an even better approximation to using initial velocities because tsunami and earthquake happen simultaneously instead of after one another.

Depth filtering is a method of approximating the actual sea surface ele-

vation when only bottom displacements are known. Kajiura's Filter is one approach to depth filtering and shows a better approximation to reality than copying displacements to the sea surface directly. Still, the effect is small and, depending on the scenario, might not be of significance. Especially regarding SWE solvers (or non-dispersive solvers in general), the resulting removal of high-frequency waves in the sea surface profile does not cause a visible difference. Furthermore it is not possible to apply Kajiura's Filter to every tsunami scenario as it was originally only developed for planar seafloors of constant depth and instantaneous displacements of it. Multiple researchers have shown it to work for real-world scenarios too and have achieved favorable results.

The most accurate initial conditions for tsunamis obtainable from SeisSol are those where the earthquake's surface waves have left the domain and where seafloor displacement, surface displacement and water velocity are available. Time-dependent displacements deliver the next best result with Kajiura's Filter applied and static seafloor displacement is a less accurate approximation but still has a comparable wave profile.

Bibliography

[1] B Andrist and V Sehr. *C++ High Performance: Boost and optimize the performance of your C++17 code*. Packt Publishing, 2018. ISBN: 9781787124776. URL: https://learning-oreilly-com.eaccess.ub.tum.de/library/view/-/9781787120952/?ar.

[2] Ivo Balbaert, Avik Sengupta, and Malcolm Sherrington. *Julia: High Performance Programming*. Packt Publishing Ltd, 2016. ISBN: 1787126102. URL: https://learning-oreilly-com.eaccess.ub.tum.de/library/view/-/9781787125704/?ar.

[3] M. Charbit. *Digital Signal and Image Processing Using MATLAB*. ISTE. Wiley, 2010. ISBN: 9780470394526.

[4] D.E. Dudgeon and R.M. Merser. *Multidimensional Digital Signal*. Prentice-Hall Signal Processing. Prentice-Hall, 1995. ISBN: 9780132276382.

[5] Kayvon Fatahalian et al. "Data-parallel rasterization of micropolygons with defocus and motion blur". In: ACM Press, 2009. DOI: 10.1145/1572769.1572780.

[6] Jorge Gascon et al. "Fast deformation of volume data using tetrahedral mesh rasterization". In: ACM Press, 2013. DOI: 10.1145/2485895.2485917.

[7] Walter Gellert et al. *The VNR concise encyclopedia of mathematics*. Springer Science & Business Media, 2012. ISBN: 9401169829.

[8] S Glimsdal et al. "Dispersion of tsunamis: does it really matter?" In: *Natural Hazards and Earth System Sciences* 13 (6 2013), pp. 1507–1526. ISSN: 1684-9981. DOI: 10.5194/nhess-13-1507-2013.

[9] Steven J Janke. *Mathematical structures for computer graphics*. John Wiley & Sons, 2014. ISBN: 1118712196. URL: https://learning.oreilly.com/library/view/mathematical-structures-for/9781118712191/.

[10] Kinjiro Kajiura. “The Leading Wave of a Tsunami”. In: 東京大學地震研究所彙報 = *Bulletin of the Earthquake Research Institute, University of Tokyo* 41 (1963), pp. 535–571. ISSN: 00408972.

[11] Lukas Krenz et al. “Elastic-Acoustic Coupling for 3D Tsunamigenic Earthquake Simulations with ADER-DG on Unstructured Tetrahedral Meshes”. en. In: *AGU Fall Meeting*. American Geophysics Union. San Francisco, USA, 2019. URL: https://agu2019fallmeeting-agu.ipostersessions.com/default.aspx?s=04-6A-5E-23-1B-D4-A6-BB-0F-24-BE-65-75-4C-03-23.

[12] Gabriel C Lotto, Gabriel Nava, and Eric M Dunham. “Should tsunami simulations include a nonzero initial horizontal velocity?” In: *Earth, Planets and Space* 69 (1 2017). ISSN: 1880-5981. DOI: 10.1186/s40623-017-0701-8.

[13] Elizabeth Madden et al. *Methods and Test Cases for Linking Physics-Based Earthquake and Tsunami Models*. 2019. DOI: 10.31223/osf.io/rzvn2.

[14] Claudio Margottini, Paolo Canuti, and Kyoji Sassa. *Landslide Science and Practice*. Vol. 1. Springer, 2013. ISBN: 978-3-642-31426-1 978-3-642-31427-8. DOI: 10.1007/978-3-642-31427-8.

[15] James R Munkres. *Elements of algebraic topology*. CRC Press, 2018, pp. 2–6. ISBN: 0429962460.

[16] Geir Kleivstul Pedersen. “A note on tsunami generation by earthquakes”. In: *Preprint series. Mechanics and Applied Mathematics* (2001). URL: http://urn.nb.no/URN:NBN:no-27814.

[17] Christian Pelties, Alice-Agnes Gabriel, and J-P Ampuero. “Verification of an ADER-DG method for complex dynamic rupture problems”. In: *Geoscientific Model Development* 3 (2014), pp. 847–866.

[18] Sebastian Rettenberger. “Scalable I/O on Modern Supercomputers for Simulations on Unstructured Meshes”. 2018. URL: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20180327-1398032-0-6.

[19] Tatsuhiko Saito and Takashi Furumura. “Three-dimensional tsunami generation simulation due to sea-bottom deformation and its interpretation based on the linear theory”. In: *Geophysical Journal International* 178 (2 Aug. 2009), pp. 877–888. ISSN: 0956540X. DOI: 10.1111/j.1365-246X.2009.04206.x. URL: https://academic.oup.com/gji/article-lookup/doi/10.1111/j.1365-246X.2009.04206.x.

[20] Hang Si. "TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator". In: *ACM Transactions on Mathematical Software* 41 (2 Feb. 2015), pp. 1–36. ISSN: 0098-3500. DOI: 10.1145/2629697. URL: https://dl.acm.org/doi/10.1145/2629697.

[21] Y Tony Song and Shin-Chan Han. "Satellite observations defying the long-held tsunami genesis theory". In: Springer, 2011, pp. 327–342.

[22] Y Tony Song, Ali Mohtat, and Solomon C Yim. "New insights on tsunami genesis and energy source". In: *Journal of Geophysical Research: Oceans* 122 (5 2017), pp. 4238–4256. ISSN: 2169-9291.

[23] Y Tony Song et al. "The role of horizontal impulses of the faulting continental slope in generating the 26 December 2004 tsunami". In: *Ocean Modelling* 20 (4 2008), pp. 362–379. ISSN: 1463-5003.

[24] Don Speray and Steve Kennon. "Volume probes: Interactive data exploration on arbitrary grids". In: *ACM SIGGRAPH Computer Graphics* 24.5 (1990), pp. 5–12.

[25] V Titov et al. "Consistent estimates of tsunami energy show promise for improved early warning". In: Springer, 2016, pp. 3863–3880.

[26] John Vince. "Barycentric Coordinates". In: *Mathematics for Computer Graphics*. London: Springer London, 2006, pp. 193–221. ISBN: 978-1-84628-283-6. DOI: 10.1007/1-84628-283-7_11.

[27] Zhigang Xu and Y Tony Song. "Combining the all-source Green's functions and the GPS-derived source functions for fast tsunami predictions - Illustrated by the March 2011 Japan tsunami". In: *Journal of Atmospheric and Oceanic Technology* 30 (7 2013), pp. 1542–1554. ISSN: 0739-0572.

Appendix A

Code Repositories

A.1 SAMPLER

The code for rasterization and for Kajiura's Filter is implemented in the SAMPLER software package found here: `https://gitlab.lrz.de/ge73tes/sampler`. The program is released as free software under the GPLv3 license.

A.2 sam(oa)^2

Modifications made to sam(oa)^2 in order for it to support initial velocities and surface displacements can be found here: `https://gitlab.lrz.de/samoa/samoa/-/tree/max-bachelor`.