Universiteit Utrecht

Large-scale Shallow Water Control

Master thesis
Game and Media Technology
ICU-3117367

Selmar Kok

First Examiner:
dr. Michael Wand

Second Examiner:
Marries van de Hoef, Msc.

External supervisor:
prof. dr. Elmar Eisemann

July 29, 2015

Abstract

The research area of water simulation control is focused on controlling the flow of computer-generated water. Current studies in this area are heavily skewed towards off-line applications. While there are studies that achieve control in real-time, in practice this is limited to small-scale scenarios. This thesis presents a simple technique to control shallow water simulations on a large scale in real-time.

The water control technique presented by this thesis works on any type of water simulation that stores its properties in a grid-based format. The technique is based on splitting the low and high frequency components of the water state through convolution with a low-pass filter. Additionally, to maintain existing water flow details, control is applied in a weak form by linearly interpolating the current water state with a target state.

We show that the technique produces similar results to an uncontrolled water simulation that is based on the same input. Furthermore, we show that important small-scale wave details of the water simulation are maintained for an extended period of time, while large-scale interruptions are quickly dealt with. Performance of the technique relies on the size of the grid, but is real-time when running on a single CPU for a typical flooding scenario of 128 by 128 cells.

Contents

1. Introduction

Water simulation is one of the largest areas of research within computer graphics. Applications for water simulation include computer games and movies, but is also useful for disaster management, to visualize for example oil spills and flooding scenarios.

Simulating a large-scale flooding scenario in the context of disaster management has one major requirement: the simulation has to be an accurate representation of a real-world flooding scenario. This means the governing equations that describe water flow, often the shallow water equations, can not be simplified much. Combined with the large scale of the simulation, this means it can not be simulated in real-time on an average desktop computer. If a real-time visualization is required, the large-scale simulation will have to be precomputed.

A real-time water flow visualization can have several degrees of realism. A realistic visualization of a flooding scenario means detailed interaction with the environment is required. For this, a water simulation is typically used. In the case of visualizing an existing large-scale simulation, it would be a small-scale water simulation on top of the large-scale simulation. One major difficulty arises: the small-scale simulation should look like it is a result of the large-scale simulation. It has to be a realistic and detailed visualization of the large-scale water flow. As the small-scale simulation will not behave like the large-scale simulation by default (it does not even know about incoming water), a way to control the small-scale simulation is required.

Objective

The research topic of this thesis is water simulation control. Specifically, it is about controlling shallow water simulations without losing the characteristics that make the water what we perceive it to be. While there are several techniques to solve this problem off-line on both small and large scales, mentioned in chapter 2, there have not yet been any to solve this in real-time on a large scale. Thus, the objective of this thesis is to present a simple technique to control large-scale shallow water simulations with coarse control data.

Controlling water flow

To apply control on a water simulation, we first need a shallow water solver. The solver we use models water velocity, water height and dry regions and includes some stability enhancements.

The focus of our water control technique is on trying to keep the details important for recognizing water. This is done by using a Gaussian high-pass filter to separate the large- and small-scale water flow. The high-frequency part will contain the small-scale water flow that we desire to keep. This filter includes some modifications to account for dry water regions. We also evaluate a problem with repeated application of a Gaussian high-pass filter.

To control the simulation, the large-scale water flow, the low-frequency part, is replaced with the 'correct' large-scale flow, taken for example from an external data source. The small-scale details are kept by adding the high- and low-frequency parts back together. Furthermore, we assume that an uncontrolled small-scale simulation and the data we use to control the simulation move roughly the same, allowing weak control on the simulation through interpolation between the current and a target water state. The end result is a robust technique to control shallow water flow.

Contributions

The technique to control shallow water flow is our main contribution. A minor contribution is an analysis of repeated high-pass filtering. Additionally, we have a very small contribution in the function that determines reflecting boundaries for our shallow water simulation.

In addition to the contributions in water simulation, we present a small advance in image-based water visualization in appendix A.

Overview

This thesis will continue with a summary of work related to our research topic in chapter 2. After that, we will give a short problem description in chapter 3. Following this, in section chapter 4, we will describe our technique from start to finish, complete with all details required to implement it. Chapter 5 will then describe the test environment and the results of the technique, including an analysis of its quality and performance. The thesis will finish with a conclusion and our future work in chapters 6 and 7.

2. Related Work

This chapter's purpose is to give an overview of the current state of the art in water simulation in section 2.1, water simulation control in section 2.2 and flow rendering in section 2.3. Each section will provide an overview of real-time techniques, if applicable, and mention relevant off-line methods otherwise.

2.1. Water Simulation

The topic of water simulation has been very well researched. At the same time, there is yet no all-round perfect technique for simulating water. As such, there are many different approaches to do so. Water simulations typically revolve around simplifying and/or solving the Navier-Stokes equations (presented in section 4.1.1) one way or another. In some cases, the techniques rely on linear wave theory, which only models water height, or the even simpler wave equations, which also assumes a constant wave speed.

2.1.1. Particle-based methods

In particle-based simulations, particles typically carry mass through space, interacting with other particles and their environment. Typical usages in real-time applications are small-scale, such as water sprays and foam.

The core of many particle-based methods is Smoothed Particle Hydrodynamics, first applied to free-surface fluid simulation by [Mon94]. It has since been extended for use in interactive applications [MCG03, KW06]. To increase performance, adaptive simulation has been implemented by [APKG07], who scale the particles, and [LTKF08], who couple SPH with a particle level-set method. Different types of fluids can also be simulated, such as viscoelastic fluids as done by [CBP].

Other particle-based methods include Langevin particles [CZY11], Moving Particle Semi-implicit(MPS) [KO96] and position-based fluids [MM13].

In contrast to the above particle simulations, where the particles carry mass, [YHK07] uses particles to carry wave information across a water surface. This has been extended for use in flowing water by [Cor08].

2.1.2. Grid-based methods

Grid-based methods are based around the idea of cells containing water. Water is then moved by cells exchanging mass and possibly other water properties like velocity and temperature. This method is very popular in real-time applications, as it is well-suited to simulating large bodies of water. It is often coupled with particle-based methods for small-scale details.

Grid-based simulation became popular with the introduction of unconditionally stable fluid models [Sta99]. Many variations have been introduced since, not only 3D or 2D, but also the 2.5D heightfield-based shallow water simulations.

3D water simulations are typically used for off-line applications. A recent advance in 3D water simulation is chimera grids by [EQYF13], who discretize space with overlapping grids that translate and rotate. This way, different (possibly moving) regions of interest can be simulated with different resolutions. A 3D / 2D hybrid has been designed by [CM11], who represent a water body's deep water with tall cells, while using a full 3D simulation for the detailed surface.

For real-time applications, shallow water simulations are very popular. They solve the shallow water equations, which are a derivation of the Navier-Stokes equations, simplified with varying assumptions, of which the most obvious is that water can be represented as a 2D surface with varying water heights. The shallow water equations are described in detail in section 4.1.2. Recent implementations of shallow water solvers have been done by for example [KP$^+$07] and [CM10]. To further improve the range of applications for shallow water simulations, several adaptive schemes have been implemented [LO07, Kal08].

Because even the shallow water equations are too demanding for real-time applications in some scenarios, there has been research into faster water simulations using linear wave theory [Tes04a, Day09] or even the wave equations [CS09]. A notable achievement is a prototype implementation in DICE's Frostbite engine by [Ott11], who approximate dispersion by simulating different wave lengths on different grids, thereby drastically reducing the number of computations per cell.

2.1.3. Other methods

Several other methods exist to simulate water, such as mesh-based simulation [CFL$^+$07], which is a position-based method, and model reduction [TLP06, WST09].

Aside from doing actual simulation, there are also procedural methods to generate animated water surfaces. One example is the Gerstner swell model, which generates wave shapes that can be summed, as implemented by [HNC02]. Another well-known and battle-tested technique generates waves given a phase/amplitude spectrum [Tes04b]. Both of these procedural methods can also be used to add details to an existing simulation. Lastly, there are also techniques to generate turbulent water flow [BHN07] or to add turbulence to an existing simulation [NSCL08, PTSG09].

2.2. Simulation Control

The goal of controlling simulations is to make the water flow in the direction the user wants, without breaking realism for the viewer (that is, it should still look and move like water). This is usually done by applying forces to the water or calculating derivatives given certain constraints in terms of water flow. Control can be applied in scenarios like a flooding hallway, but could also be used to create an animated figure made of water.

Research in this area typically only includes off-line applications, with some exceptions like [PHT$^+$13], which is interactive only on a small scale. Many off-line simulation control methods use the adjoint method to calculate the derivatives required to steer the simulation in the correct direction, usually towards a target keyframe [MTPS04]. [TKPR09] use control particles to apply force-based control. Additionally, similar to our method, they split the low and high frequencies of the velocity to be able to apply control only to the large-scale flow. [NB11] use a guide shape, limiting their control computations to a thin layer around it. Similarly, [RTWT12] use a mesh as their target, but control their simulation more strictly by keeping the mass constant.

2.3. Flow Rendering

Instead of simulating and controlling water, the alternative is to render the water flow in a simpler way. [YNBH09] does this by moving sprites with the flow on a flowing river. In a more 'Eulerian' approach, [vH11] splits the domain in overlapping squares, rendering water textures on them and moving the textures in the direction of the flow.

An a slightly different approach, some have tried to advect the pixels in an image with the flow, trying to maintain the texture [VW02, YNBH11]. A similar technique has also been applied in games [Vla10], where they simply keep the texture by blending the 'old' advected image with a fresh image that still has its texture.

3. Problem Description

The goal of this project is to investigate solutions to render large-scale water simulations in real-time. In our case, the result is intended to be integrated in a flood visualization framework. This software can be used to predict the flooding impact and has the potential to ease decision-making in the context of disaster management.

The large-scale water simulation the software takes input from exists only on a very coarse scale and can spread to over tens of square kilometres. In order to visualize this simulation while maintaining realism, the existing simulation will have to be refined. This could be done, for example, by projecting the coarse data on a higher resolution grid and to interpolate what lies in between.

Initially, image-based rendering might seem like a good solution. However, image-based water simply can not realistically display (violent) water flow over a rough surface. This is a typical flooding scenario. Observe for example the water in the video located at `https://www.youtube.com/watch?v=_VD5GxluHN8`. It is also very difficult for an image-based water renderer to interact with the bottom elevation to create things like whirlpools.

The goal of this thesis is to find a solution for displaying turbulent water flows within the constraints of the given simulation data. This solution is presented in chapter 4.

4. Model

This chapter defines a solution for rendering detailed water flow under constraints given by a simulation performed on a larger scale. We will provide all the details required to reproduce the result, as well as give arguments for the chosen sub-methods and parameters.

First we pose a global overview of our method with algorithm 1. We will then explain some techniques and algorithms we use in our method in sections 4.1 to 4.3. Finally, we will describe the algorithm in detail and how we put it all together in section 4.4.

From here on out, unless mentioned otherwise, *small-scale data* will refer to the small-scale water simulation or state. Similarly, *control data* or *large-scale data* refers to the data used to control this simulation. The latter can refer to the data interpolated over time and space, only over space or not at all, depending on context. Ambiguities will be explained. Similarly, *water state* refers to the combination of water depth/height, x-velocity and y-velocity. It will also be referred to as $S(x, y, t)$, $S(x, y)$, S_t or just S, where x and y are the x- and y-position in meters respectively and t is the frame time in seconds. It can also be referred to as $S(i, j)$, meaning it is indexed with grid indices instead of position. It refers to the small-scale data by default. When S refers to the large-scale data, it will be denoted with $_L$.

Algorithm 1 Main loop (executed every frame)

```
1: Interpolate control data over space and time. (section 4.4.3)
2: Extract high-frequency data from water state using Gaussian filter. (sections 4.2 and 4.3)
3: Mix current water state with high-frequency data and control data. (section 4.4.1)
4: Apply shallow water simulation algorithm to water state. (section 4.1)
```

The cell width and/or time step of the large-scale data is larger than that of the small-scale simulation. As such, the control data, which exists only on a large scale, will have to be interpolated first. We assume there is too much data to do this beforehand, so this is for practical reasons like memory usage and disk space.

Next, existing high-frequency data is extracted from the current water state. This is done with a Gaussian filter. The idea is to replace the low-frequency data with something we want it to be, in this case the interpolated large-scale data.

Then, the current water state is mixed with the obtained high-frequency and control data. We use a level set method to determine the free-surface boundary region of the large scale data. The current water state is treated differently depending on whether it is outside of, inside of or on the boundary.

4.1. Shallow Water Simulation

To describe motions of a fluid like water, usually, the Navier-Stokes equations are used (section 4.1.1). These equations describe the 3D motion of a fluid. Since computing large-scale 3D water flow on a regular PC in real-time is difficult, we have chosen to use the simpler shallow water equations (SWE, section 4.1.2). The SWE reduce the problem to 2D space, using a heightfield to represent the third dimension. We believe they give the best trade-off between realism and performance, while remaining relatively simple to implement. Unlike simpler pipe models, they still allow for vortices and take the dispersion relation into account which links maximum wave speed to water depth. However, any water simulation algorithm can be used, as long as it stores the water state in a similar manner. The algorithm described in this thesis is a combination of the methods used in [CM10] and [MSJT08]. Only using the algorithm described in [MSJT08] will give problems with mass conservation.

From here on we use the following notation, mostly following the conventions from [CM10][1]. For our units, we use meters and seconds.

- h is the depth of the water,
- H is the height of the bottom elevation,
- $\eta = H + h$ is the height of the water above zero-level (see figure 4.1a),
- $\vec{v} = (u, v)^\intercal$ is the horizontal velocity of the water,
- g is gravity,
- t is time.

We define the water state as follows:

$$S_t = S(x, y, t) = \begin{pmatrix} \eta(x, y, t) \\ u(x, y, t) \\ v(x, y, t) \end{pmatrix} = \begin{pmatrix} \eta_t \\ u_t \\ v_t \end{pmatrix} \tag{4.1}$$

4.1.1. Navier-Stokes Equations

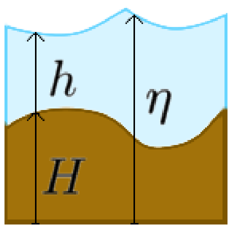
The incompressible Navier-Stokes equations can be written as follows [bri]:

$$\frac{\delta \vec{v}}{\delta t} + \vec{v} \cdot \nabla \vec{v} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{v} \tag{4.2a}$$

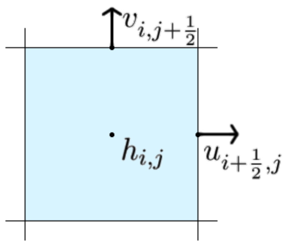
$$\nabla \cdot \vec{v} = 0 \tag{4.2b}$$

Here, ρ is the fluid density, p is the pressure, $\vec{g}$ represents external forces such as gravity and ν is the kinematic viscosity.

[1]The exceptions are the symbols for the velocity. Also, it is worth noting that [MSJT08] uses significantly different conventions.



(a) Elevations



(b) Staggered grid

Figure 4.1.: Spatial layout of water state.

4.1.2. Shallow Water Equations

The shallow water equations are derived from the Navier-Stokes equations. See [MSJT08] for a detailed derivation. Several assumptions are made in order to simplify the previously mentioned Navier-Stokes equations:

- The water pressure is hydrostatic, which implies the vertical velocity is temporally constant and equal to 0.
- The density of the water is constant.
- The viscosity of the water is equal to 0.

The shallow water equations can be written as follows:

$$\frac{Dh}{Dt} = -h\nabla \cdot \vec{v} \tag{4.3a}$$

$$\frac{D\vec{v}}{Dt} = -g\nabla\eta \tag{4.3b}$$

Here, D is the material derivative operator, $\frac{Dx}{Dt} = \frac{\delta x}{\delta t} + \vec{v} \cdot \nabla x$, where x is a property of the material such as water height or temperature. It can be seen as a derivative that follows the motion of the water [BSL07]. In practice, this means we not only have to integrate the formulas, but also advect (move) x with the velocity of the water.

4.1.3. Algorithm

Before we can do any integration, we have to discretize our domain. As is common in fluid simulation, we use a uniform grid to represent the height of the water, with the velocities stored on the boundaries of the cells. See figure 4.1b. We define our domain to be a grid of n_1 by n_2 points. We use the zero-based indices i and j respectively to index into this grid. Unless mentioned otherwise, an equation like $h_{i,j} \mathrel{+}= \frac{\delta h_{i,j}}{\delta t}\Delta t$ is meant to be applied to every combination of the indices i and j. When we index velocities, we use for example $u_{i+\frac{1}{2},j}$ to indicate the value of u stored on the right border of $h_{i,j}$ and $v_{i,j-\frac{1}{2}}$ for the value of v on the lower border of $h_{i,j}$. Δx is the cell width or grid spacing and Δt is the time step size.

All steps in the shallow water solver algorithm 2 are applied on the whole grid, unless mentioned otherwise. When implementing any of the steps, keep in mind that some of the variable state updates rely on themselves (ie. u when advecting u), so these variable states (or the results) need to be cached or the results might not be correct.

Algorithm 2 Advection

```
1: Advect $u$
2: Advect $v$
3: Advect & Integrate $h$
4: Set domain boundaries
5: Set water level, $\eta = H + h$
6: Integrate $u$
7: Integrate $v$
```

Velocity advection

The advection algorithm used is taken from [MSJT08], which uses the unconditionally stable method from [Sta99]. See equation (4.4). It performs advection by projecting backwards in time. Given a position on our grid, $\mathbf{x}$, we trace an imaginary particle back in time to get to a position $\mathbf{x}'$. We set the property we are advecting, s, at $\mathbf{x}$, to the value of the property found at $\mathbf{x}'$.

Since $\mathbf{x}'$ can be outside of the simulation domain, it should be clamped to stay on the grid. Additionally, we have to perform bilinear interpolation at $\mathbf{x}'$, because we will usually end up between the grid points where the values are stored.

The staggered grid stores the u and v parts of the velocity in different locations, so we have to advect twice, once for u and once for v. They require different interpolations to get a complete velocity vector, so they have to be advected separately. Simple averaging can be used, as it is equal to bilinear interpolation in those cases.

$$x'_{i,j} = x_{i,j} - \Delta t \cdot \vec{v}_{i,j} \tag{4.4a}$$

$$s_{i,j} = \mathit{bilinearInterpolation}(s, x'_{i,j}) \tag{4.4b}$$

Height integration

Instead of explicitly advecting h, we use the method from [CM10], which combines the height advection step with the height integration. This way, we can guarantee mass conservation. According to them, equation (4.3a) can be rewritten to:

$$\frac{\delta h}{\delta t} = -\nabla \cdot (h\vec{v}) \tag{4.5}$$

After discretization, their result is:

$$\frac{\delta h_{i,j}}{\delta t} = -\left(\frac{(\bar{h}u)_{i+\frac{1}{2},j} - (\bar{h}u)_{i-\frac{1}{2},j}}{\Delta x} + \frac{(\bar{h}v)_{i,j+\frac{1}{2}} - (\bar{h}v)_{i,j-\frac{1}{2}}}{\Delta x}\right) \tag{4.6}$$

Here, $\bar{h}$ is h evaluated in the upwind direction, for example:

$$\bar{h}_{i+\frac{1}{2},j} = \begin{cases} h_{i+1,j} & \text{if } u_{i+\frac{1}{2},j} \leq 0 \\ h_{i,j} & \text{if } u_{i+\frac{1}{2},j} > 0 \end{cases} \tag{4.7a}$$

$$\bar{h}_{i,j-\frac{1}{2}} = \begin{cases} h_{i,j} & \text{if } v_{i,j-\frac{1}{2}} \leq 0 \\ h_{i-1,j} & \text{if } v_{i,j-\frac{1}{2}} > 0 \end{cases} \tag{4.7b}$$

It is then explicitly integrated:

$$h_{i,j} \mathrel{+}= \frac{\delta h_{i,j}}{\delta t}\Delta t \tag{4.8}$$

Velocity integration

The velocity integration step is equal for both [CM10] and [MSJT08].

$$u_{i+\frac{1}{2},j} \mathrel{+}= -g\frac{\eta_{i+1,j} - \eta_{i,j}}{\Delta x}\Delta t \tag{4.9a}$$

$$v_{i,j+\frac{1}{2}} \mathrel{+}= -g\frac{\eta_{i+1,j} - \eta_{i,j}}{\Delta x}\Delta t \tag{4.9b}$$

Boundary conditions

We use the same reflecting domain and free-surface boundary conditions as [CM10]. The values of η on the domain boundary are always set to be equal to their closest non-boundary neighbour. In addition, the values of the velocities on the borders of this boundary should be set to 0 and never updated during the velocity advection or integration steps.

For the free-surface boundary conditions, we say a cell is dry when equation (4.10) is true. A face $(i + \frac{1}{2}, j)$ is reflective when either equation (4.11a) or equation (4.11b) is true. This is similar for a face $(i, j + \frac{1}{2})$, every $_{i+1,j}$ becomes $_{i,j+1}$. Reflective faces have their corresponding velocity value set to 0 during the velocity integration step. In words, a face is reflective if two sides are dry or if one side is dry and the other side has water that can flow into the dry cell. We made a slight modification to the equations used by [CM10]: they use $H > \eta$, we use $H \geq \eta$. This way faces between dry cells of equal height are also properly marked as reflective.

$$h \leq 10^{-4}\Delta x \tag{4.10}$$

$$h_{i,j} \leq 10^{-4}\Delta x \wedge H_{i,j} \geq \eta_{i+1,j} \tag{4.11a}$$

$$h_{i+1,j} \leq 10^{-4}\Delta x \wedge H_{i+1,j} \geq \eta_{i,j} \tag{4.11b}$$

Stability enhancements

The authors of [CM10] add a few stability enhancements. They mention that due to numerical error, h can become smaller than zero. This can cause stability issues. We have not observed this with the standard solver, but since we are going to modify the water state later on we adapt their method of clamping h to be ≥ 0. We do this in the height integration step. Additionally, in the velocity integration step, we clamp the magnitudes of u and v to be less than $\alpha\frac{\Delta x}{\Delta t}$. We used $\alpha = 0.45$ instead of 0.5 as [CM10] do, because of an issue related to velocity advection, causing water flow to almost come to a halt. In hindsight, this might not have been necessary, as later testing did not show any obvious differences. The issue was likely related to erroneous bilinear interpolation in the code that was fixed in the meantime. The end result is not much different, as the difference is relatively small and clamping is a rare event to begin with, which happens only with very large waves and steep slopes.

4.2. Gaussian Low-pass Filter

When controlling the shallow water simulation, we want to separate the large-scale flow from the small-scale details. To achieve this, we apply a Gaussian low-pass filter to extract the large-scale (low-frequency) part.

This section describes the details for the process of separating the high and low frequencies of our water state through convolution with a Gaussian filter kernel. We start with describing the convolution process in section 4.2.1, followed by a modification of it in section 4.2.2. After that we will describe the Gaussian function as a filter and the filter kernel itself in sections 4.2.3 and 4.2.4.

4.2.1. Convolution

The 1D and 2D convolution operations are defined by equation (4.12), where f and g are 2d images. x and y are discrete variables, the grid indices of the pixel we are currently operating on. n_1 and n_2 are half the extents of g in x- and y-direction respectively. In our case, f is a grid that stores a property of the water state like h. g is the filter kernel, where n_1 and n_2 are half the kernel size, rounded down. Note that the kernel has to have uneven sizes for this to work. The Gaussian is a separable filter, so we perform the 2D convolution in two consecutive 1D passes, once for x, once for y.

$$g(x,y) * f(x,y) = \sum_{i=-n_1}^{n_1} \sum_{j=-n_2}^{n_2} g(i,j) f(x-i, y-j) \tag{4.12a}$$

$$g(x) * f(x) = \sum_{i=-n_1}^{n_1} g(i) f(x-i) \tag{4.12b}$$

4.2.2. Filter masking

Because water height is stored as bottom elevation plus water depth, using the Gaussian filter as-is can cause problems. For dry regions, we would be filtering the bottom elevation instead of the surface. To solve this, we introduce a type of masking for the convolution operation. What we mask specifically is discussed in section 4.4.5.

Since our filter is a separable one, we will present our modifications in 1D. We modify equation (4.12b) to equation (4.13), where m is the grid that stores the mask values. $m(x)$ can be 0 (*masked*) or 1 (*not masked*). In the code, we first check if $m(x)$ is 0 before doing any convolution for the current pixel, thus avoiding the possible division by zero.

$$g(x) * f(x) = m(x)\frac{\sum\limits_{i=-n_1}^{n_1} g(i)m(x-i)f(x-i)}{\sum\limits_{i=-n_1}^{n_1} g(i)m(x-i)} \tag{4.13a}$$

$$m(x) = \begin{cases} 0 & \text{if } x \text{ is } \textit{masked} \\ 1 & \text{if } x \text{ is } \textit{not masked} \end{cases} \tag{4.13b}$$

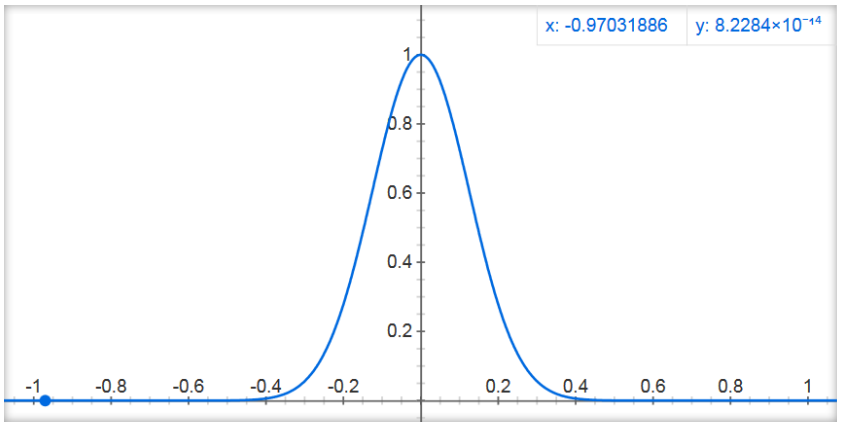
4.2.3. Gaussian filter

We chose a Gaussian filter to filter out low-frequency details of the water. There are several reasons for this:

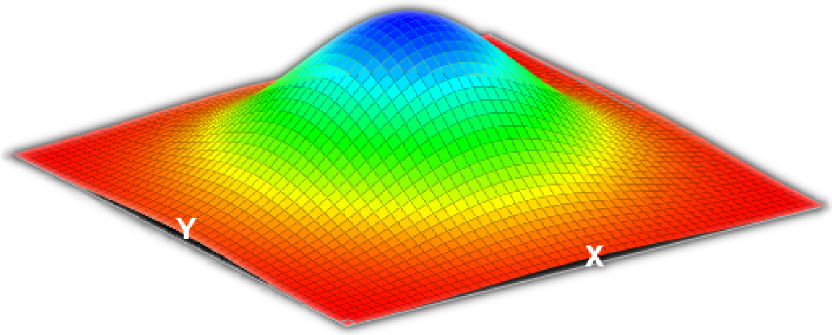
- The Gaussian has a decent spatial response. It is a simple blur. There should be no unexpected or strange artifacts resulting from this property.
- The frequency response of a Gaussian is also a Gaussian [AGJN02]. Again, this should make the result predictable.
- It is rotationally symmetric [JKS95]. This is important, because we do not want to emphasize signals going in any specific direction.
- The Gaussian filter is separable [JKS95], which means it can be implemented in 2D as the product of two 1D filters, one for each direction. This is efficient and simple to implement. Even if this is not efficient enough, there has been research into faster implementations, for example with box filters [BETVG08] or via integral images [Kov10].

A downside of the frequency response being a Gaussian is that the filter is far from ideal. Since a Gaussian is never 0, the entire frequency spectrum is affected, even though there is a clear bias towards the lower end of the spectrum. In practice, with repeated application, this means that eventually every frequency (except for 0) will be filtered out completely.

A filter is applied by convolving the original signal with the filter function [S$^+$97]. The original signal in this case is the water state $S(x, y)$. Each part of the water state is convolved separately. The 1D and 2D Gaussian functions $g(x)$ and $g(x, y)$ are given by



(a) 1D Gaussian with $\sigma = 0.125$



(b) 2D Gaussian

Figure 4.2.

equations (4.14a) and (4.14b) respectively, where x and y are spatial dimensions and σ is the standard deviation. See figure 4.2 for a visual representation.

$$g(x) = \exp(-\frac{x^2}{2\sigma^2}) \tag{4.14a}$$

$$g(x,y) = \exp(-\frac{x^2+y^2}{2\sigma^2}) \tag{4.14b}$$

The relationship between the standard deviation of the time domain, σ_t, and frequency domain, σ_f, is given by equation (4.15) [S+97].

$$2\pi\sigma_f = \frac{1}{\sigma_t} \tag{4.15a}$$

$$\sigma_t = \frac{1}{2\pi\sigma_f} \tag{4.15b}$$

4.2.4. Gaussian kernel

The kernel will be defined on the scale of the small-scale grid. This means a standard deviation of 1 in the spatial domain is exactly the Δx from the small-scale shallow water simulation. Since we implement a separable filter, we will calculate the parameters of the kernel for the 1D function only. The kernel can then be filled with values sampled from the resulting Gaussian function, placed on the center of the kernel.

We define the cut-off frequency of our filter, f_c, to be equal to the standard deviation in the frequency domain, which relation is given by equation (4.15). We can then determine an appropriate standard deviation for the Gaussian:

$$\sigma_t = 1/(2\pi f_c) \tag{4.16}$$

The frequency response of the Gaussian filter $\hat{g}(f)$ at this cut-off frequency equals:

$$\begin{aligned} \hat{g}(f) &= \exp(-f_c^2/2\sigma_f^2) && (4.17\text{a}) \\ &= \exp(-\sigma_f^2/2\sigma_f^2) && (4.17\text{b}) \\ &= \exp(-\tfrac{1}{2}) \approx 0.607 && (4.17\text{c}) \end{aligned}$$

Since the Gaussian has a transition bandwidth that spans the entire frequency band $\langle 0, 0.5]$, it is difficult to determine a proper cut-off frequency. Initially, we will start with equation (4.18), where *ControlScale* is the scale of the large scale simulation compared to the small scale simulation. This arbitrary decision is related to the Nyquist frequency. The sample rate of the large-scale grid is $1/\mathit{ControlScale}$, which means the highest frequency we can extract will be $1/(2 * \mathit{ControlScale})$. This would be the highest frequency the large grid can represent. Thus, we should at least try to filter out the higher frequencies as good as we can; they are the important details we want to keep. In our experiments, $\mathit{ControlScale} = 8$.

$$f_c = 1/(2 * \mathit{ControlScale}) \tag{4.18}$$

The resulting standard deviation is used to calculate the Gaussian kernel. The size of this kernel is given by equation (4.19), taken from [Tur07]. It is rounded down before multiplying with 2, because the kernel size needs to be an uneven number for the convolution process. After calculating the kernel, it is normalized by dividing each value by the accumulated values of the kernel.

$$\mathit{KernelSize} = 2 * \lfloor 3 * \sigma \rfloor + 1 \tag{4.19}$$

In our experiments, we will vary the cut-off frequency to determine its effect. We expect the optimal value to be smaller, where the higher frequencies are damped less. In contrast, we also expect the margin of acceptable values to be relatively wide.

4.3. Gaussian High-pass Filter

We use the low-pass filter to extract low-frequency data. The complement of this is the high-frequency data, the details we want to preserve over time. Thus, our high-pass filter at a certain point in time is given by equation (4.20), where g is a low-pass filter and g^{c} is a high-pass filter.

$$g^{\mathsf{c}}(S_t) = S_t - g(S_t) \tag{4.20}$$

The next section is about artifacts related to the Gaussian high-pass filter. It is not required for replication of our technique.

4.3.1. Gaussian high-pass filter artifacts

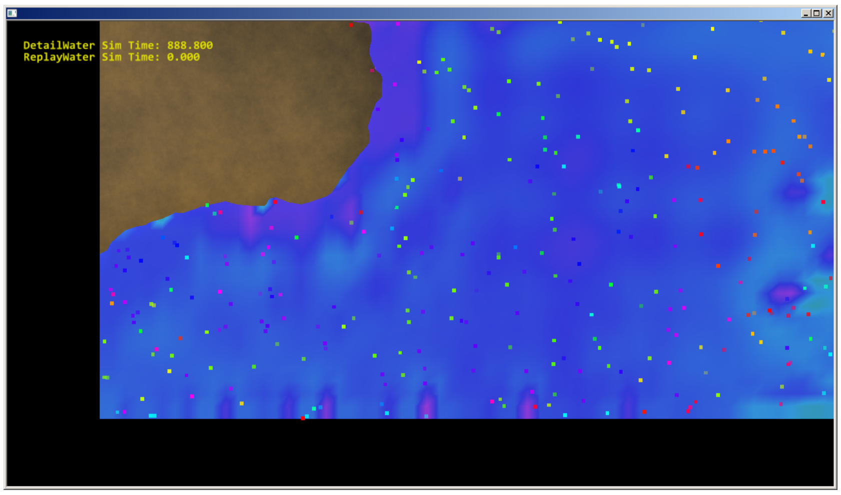
After implementing the Gaussian filter, strange artifacts started appearing several minutes into the simulation. Instead of calm, almost still water, mass clumps up in certain locations and the velocities behave strangely. See figures 4.3a and 4.3b. The large colored area is the water, which is colored with an hsv scheme depending on water height. The small squares in the image are particles that move with the velocity of the water, colored with an hsv color scheme depending on the angle of the velocity. Figure 4.3c shows the colors for the main directions of the particles. They are colored with a hue scheme that relies on the angle relative to the horizontal x-axis.

In the first image, there are particles moving up (green) and particles moving down (purple, difficult to see) right next to each other. A similar thing happens in the second image, but for horizontal directions. In order to find out the cause, we set up a 1D testing environment for the high-pass filter.

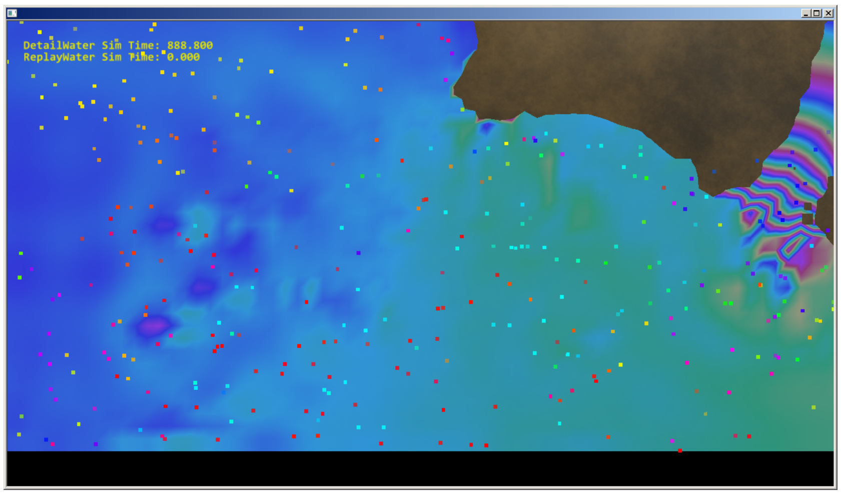
Signal responses

The first test was to apply the high-pass filter on a step function. Figure 4.4 shows how it evolves over multiple iterations. We also tested it on a Gaussian function (figure 4.5). We initially perform these filter applications on a low-resolution graph with a kernel size equal to our own usage. The graph resolution is 100 data points (x-axis), with wrapping boundaries. Green is the original signal, red is the low-pass filter and blue is the high-pass filter. This will be the case for all graphs in this section.

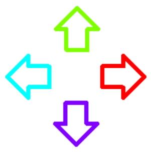
Testing showed that, after repeated filtering, the graph eventually shows numerical instability if any kind of non-zero signal was introduced. The Gaussian function is initially almost entirely eliminated from the signal by the fourth iteration (the largest value is approx. $3.5 \cdot 10^{-4}$). Regardless, after a sufficiently large number of iterations, the simulation shows instabilities. Figure 4.5b shows an iteration where the artifacts have just began growing. Introducing a little noise anywhere in the initial Gaussian causes it to happen faster.



(a) Vertically moving artifacts.

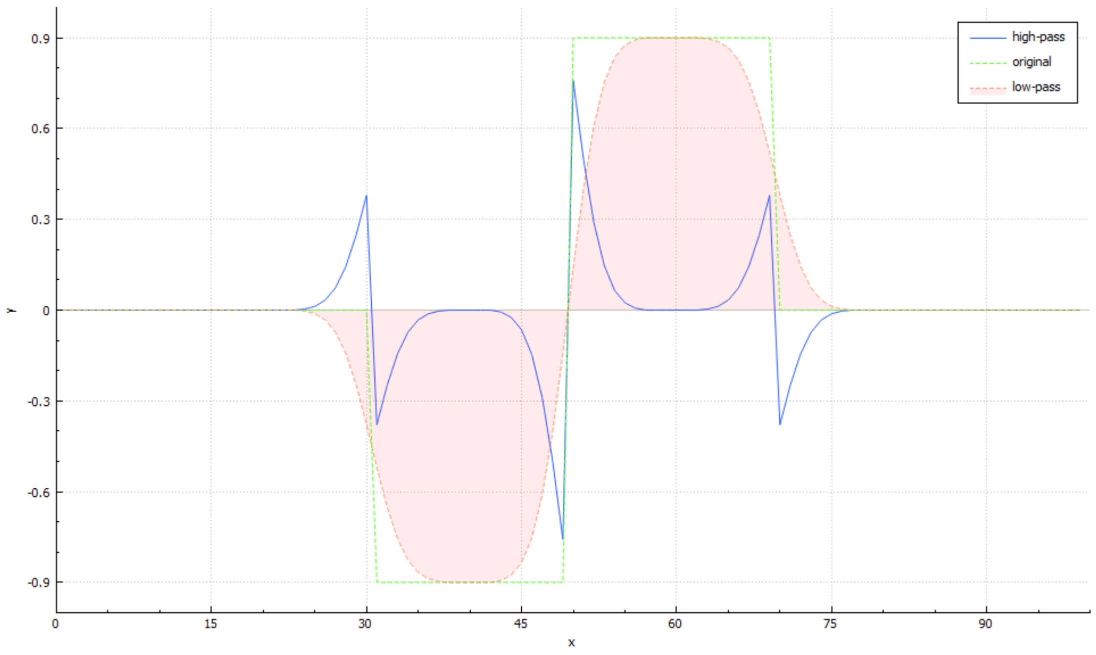


(b) Horizontally moving artifacts.

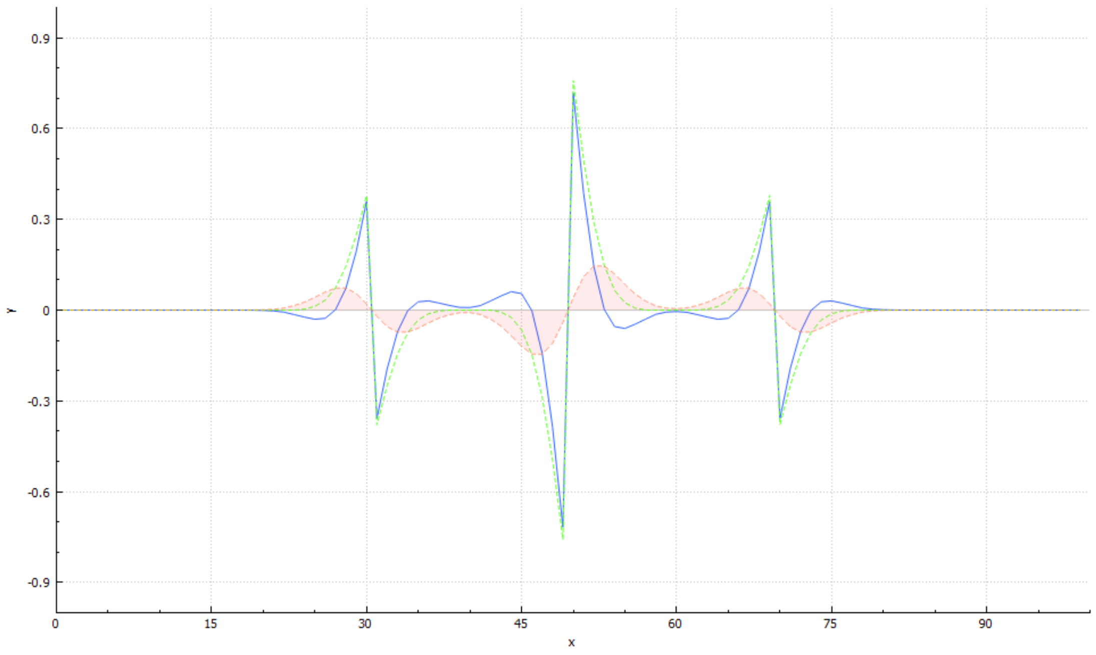


(c) Colors of particle directions.

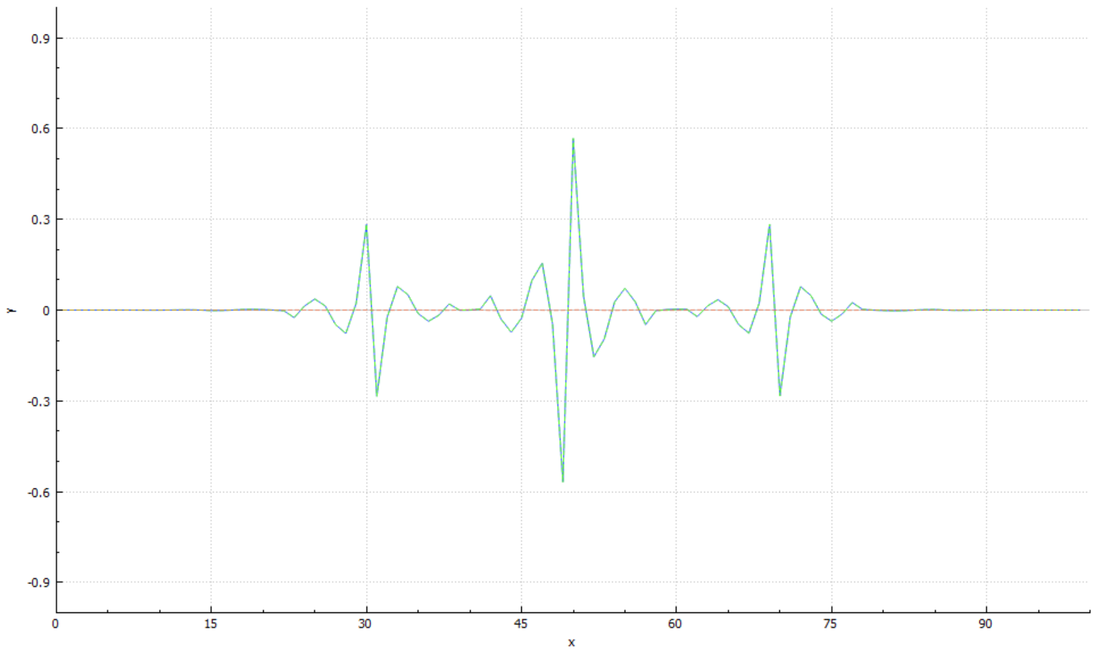
Figure 4.3.: High-pass filter artifacts.



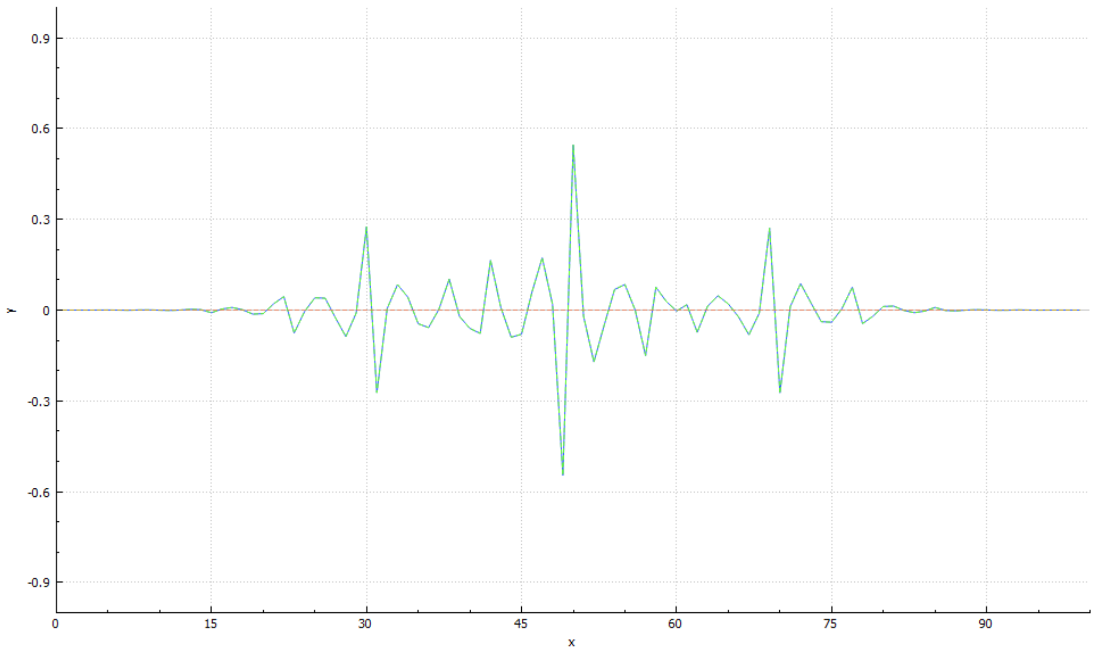
(a) First iteration.



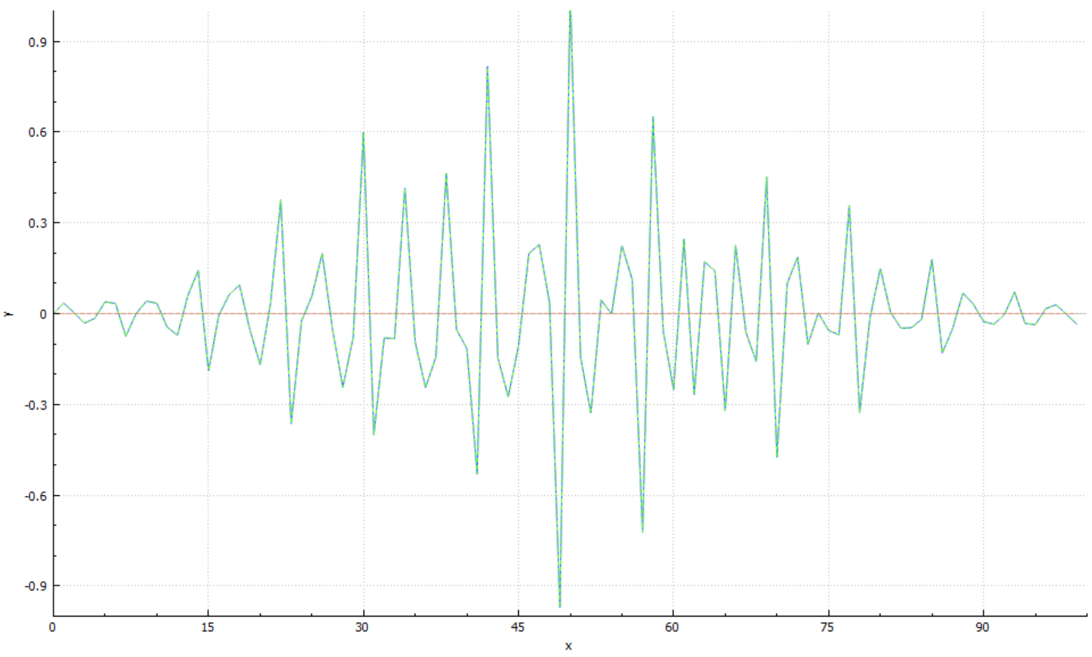
(b) Second iteration.



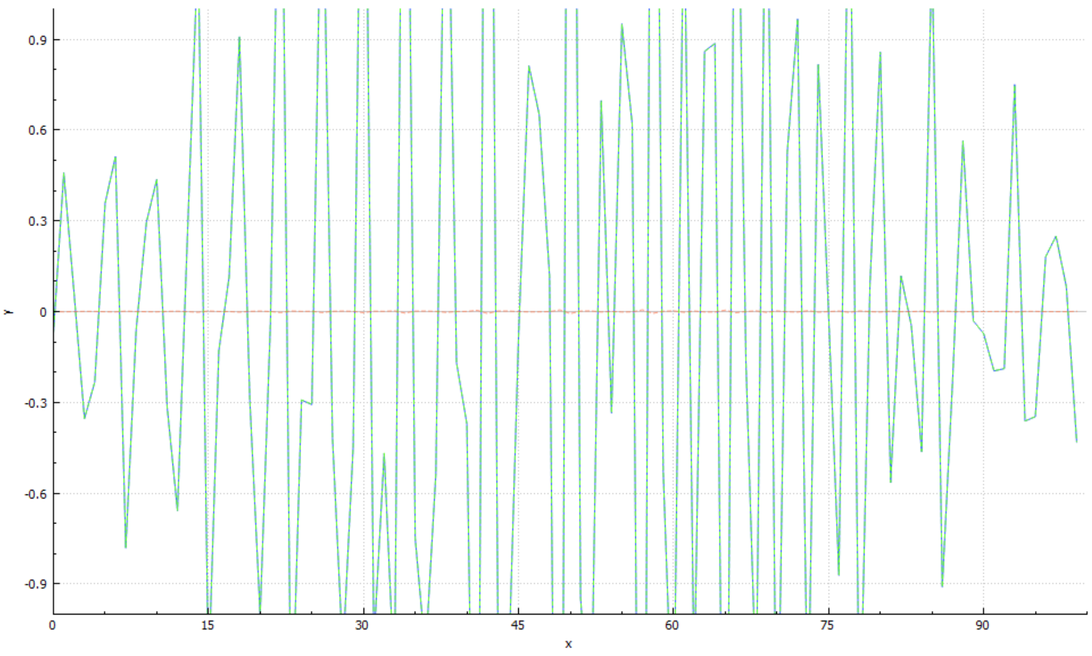
(c) Fiftieth iteration.



(d) Two hundredth iteration.

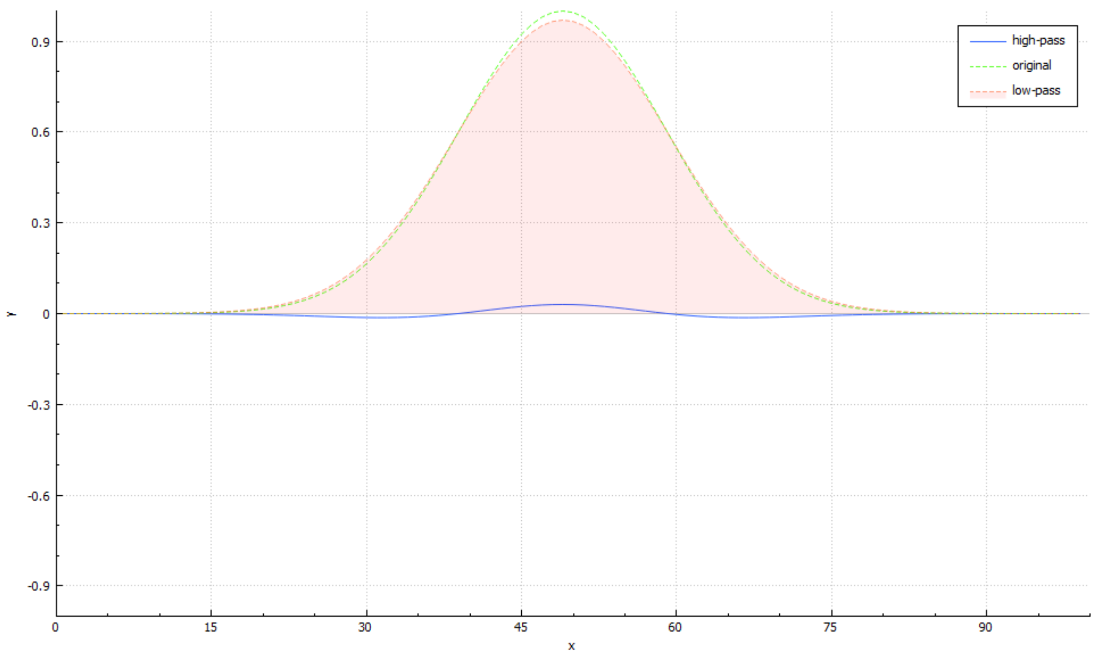


(e) Thousandth iteration.

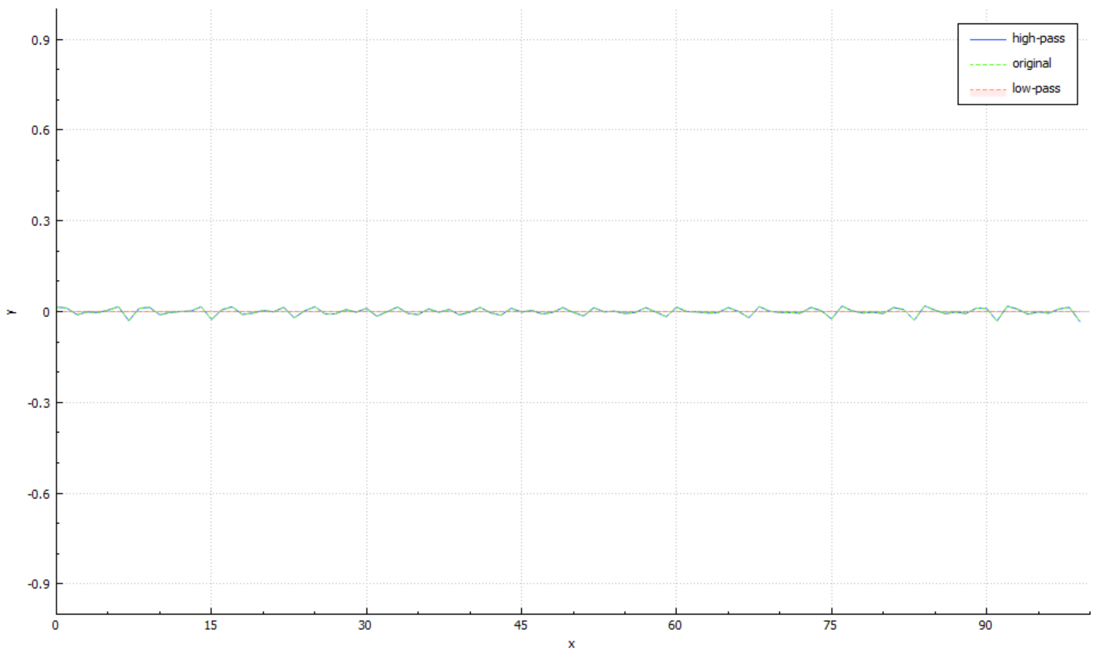


(f) Two thousandth iteration.

Figure 4.4.: High-pass filter artifacts on a simple step function.



(a) First iteration.



(b) Seven thousandth iteration.

Figure 4.5.: High-pass filter artifacts on a Gaussian function.

Discretization

It was noticed, during the water simulation, that a wider kernel causes the artifacts to appear later or disappear entirely. This lead us to the hypothesis that these artifacts may arise due to the non-continuous nature of the signal and/or kernel. Testing in our custom 1D environment shows, however, that with a higher resolution the same artifacts still arise. For the resolution increase from 100 to 1000 (meaning 1000 data points instead of 100), they do appear somewhat later. A resolution of 10000 gives a graph evolution very similar to that of 1000. See figure 4.6. Taking into account the fact that the simulation runs 20 frames per second, the actual difference in time at which the artifacts should appear is almost negligible. The reason for their slower appearance in the simulation is likely due to the self-damping of the simulation, which amplifies the delay.

Floating point accuracy

Since the values resulting from the low-pass filter can get very small and floating point accuracy is highest here, another hypothesis was that maybe subtracting or adding something so small from a higher value might in reality subtract or add slightly more than intended, causing a gradual increase in some parts of the signal. If this was the case, changing from 32-bit floating point to 64-bit floating point numbers should at least slow the appearance of the artifacts. However, it gives the exact same graphs as a result, so this could be ruled out.

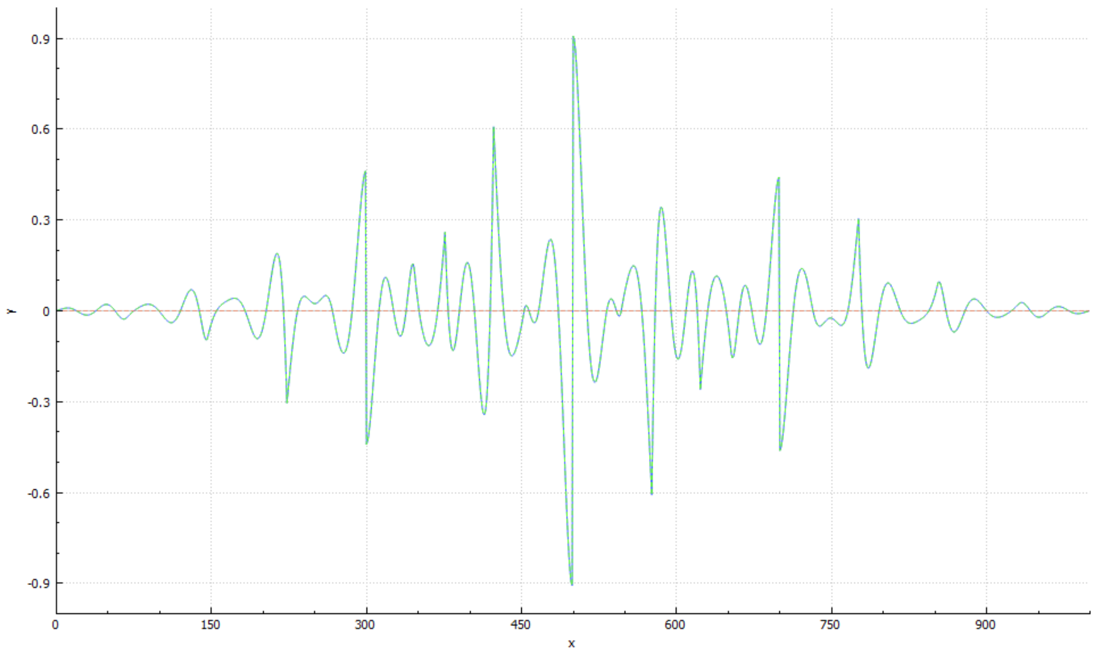
Gaussian width

The structure of the artifacts is related to the standard deviation of the applied Gaussian filter. This is easy to see in figure 4.7.

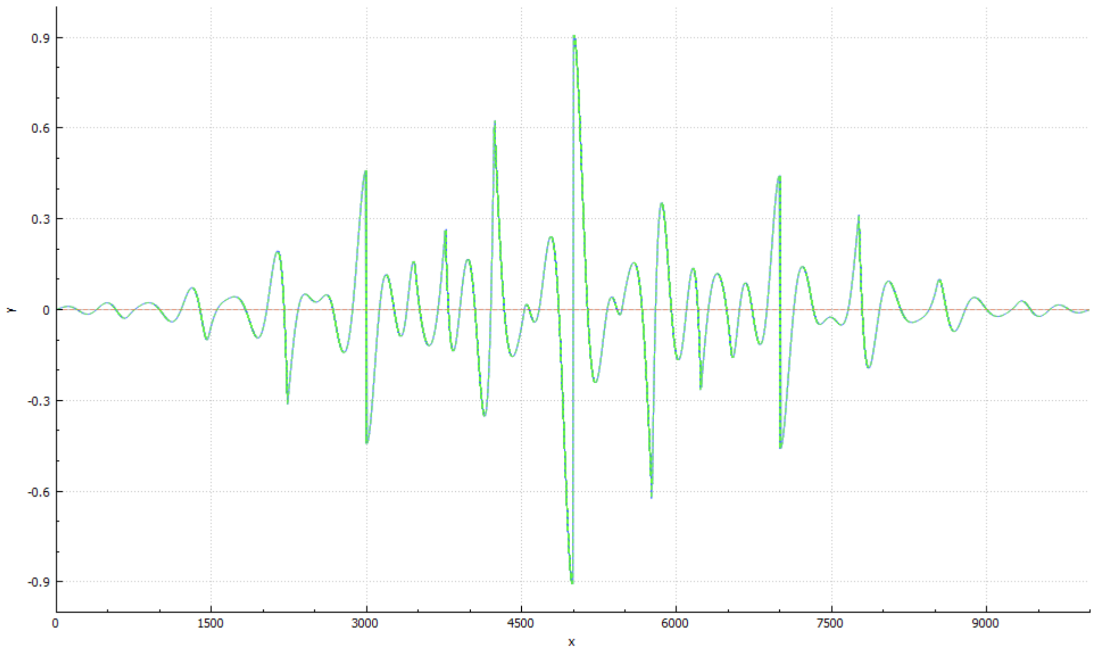
Further investigation

After ruling out other possibilities, the problem had to lie with the Gaussian filter itself. On this topic, some information is mentioned in [Luc14, ch. 12]. The subject is on waveform filtering. It mentions that all filters produce distortions, but that "distortions produced by the high-pass filter may be more problematic because they may lead to the appearance of artifactual peaks". They mention Gaussian filters do this. The results show indications that such peaks are created and are slowly amplified after many iterations.

To verify the problem lies with the Gaussian filter, we implemented a sinc filter with a blackman window [S^+97, ch. 16]. Testing in the 1D environment (figure 4.8) shows the windowed sinc causes the data in the graph to show instabilities around 10 times slower. The peaks are a little higher initially and that goes for both the correct and artifactual peaks that lie in between. Notice also the stronger oscillatory movement at the left and right tails of the signal. In the actual simulation environment, we noticed significantly more oscillations. Due to time constraints, finding out if the windowed sinc is better or if there are even better filters is postponed to future work.

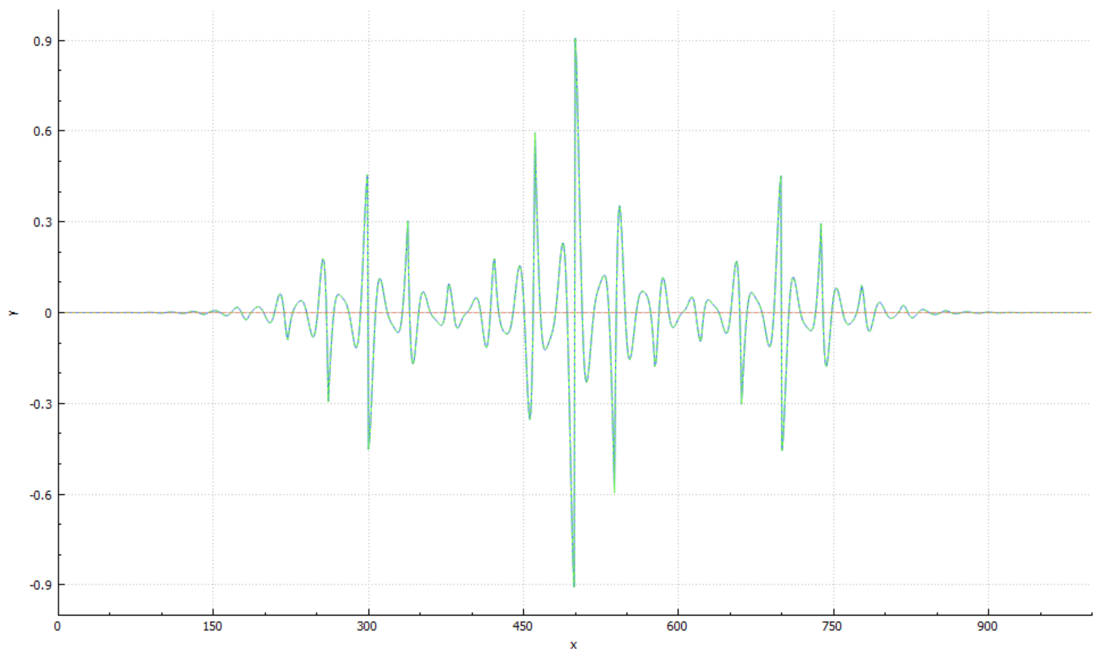


(a) Thousandth iteration, resolution 1000.

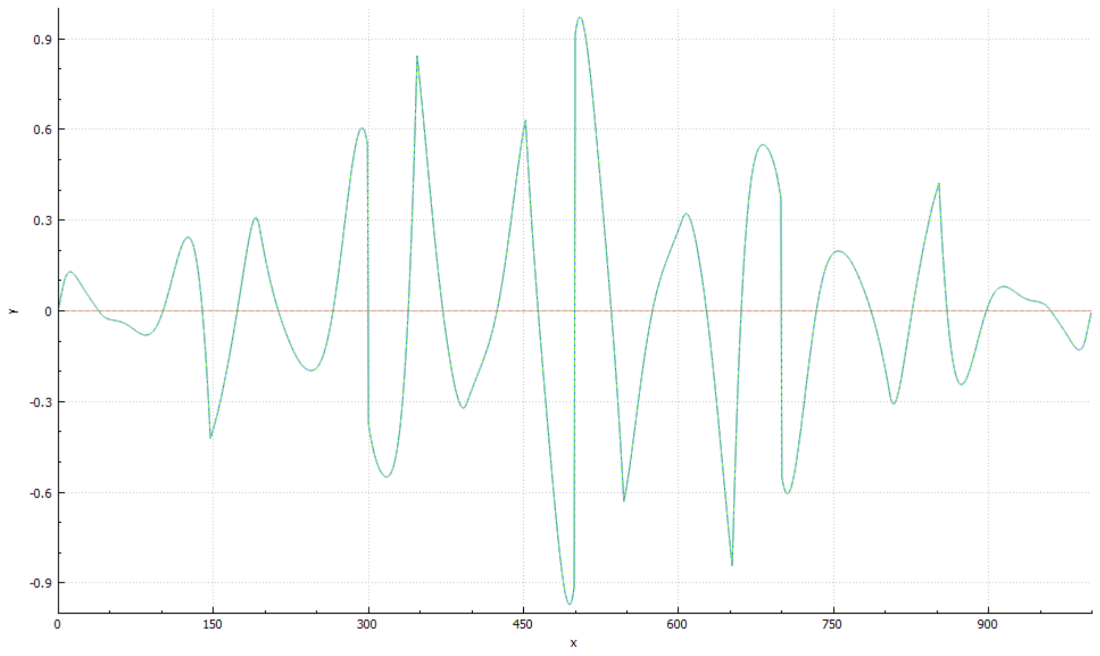


(b) Thousandth iteration, resolution 10000.

Figure 4.6.: High-pass filter artifacts on a step function with different resolutions.

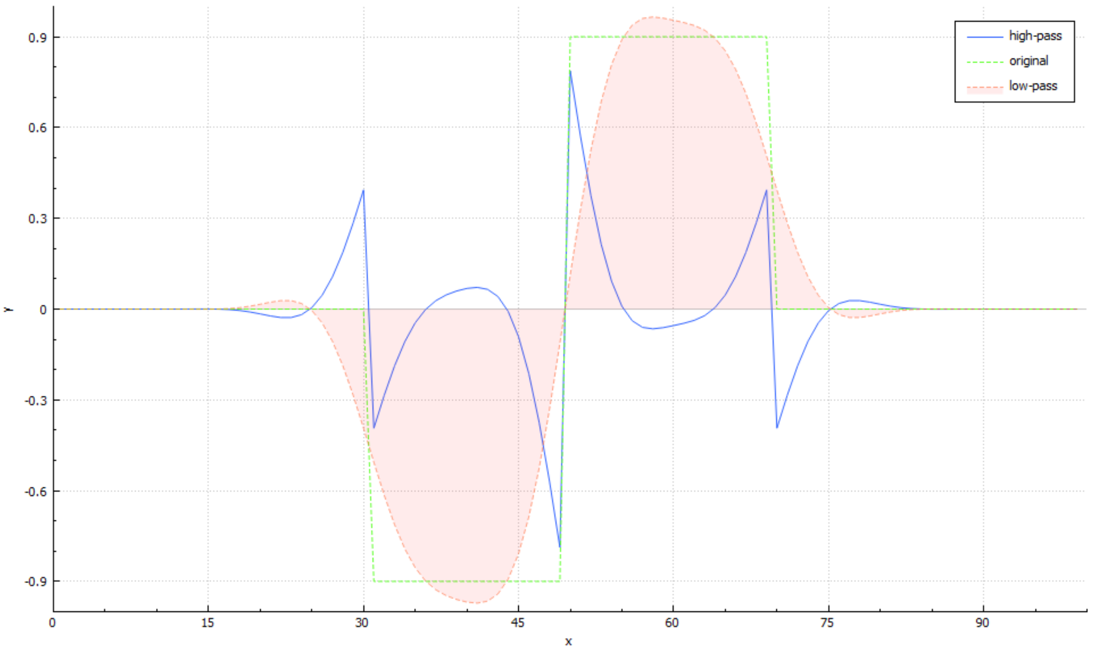


(a) Thousandth iteration, halved standard deviation for the filter kernel.

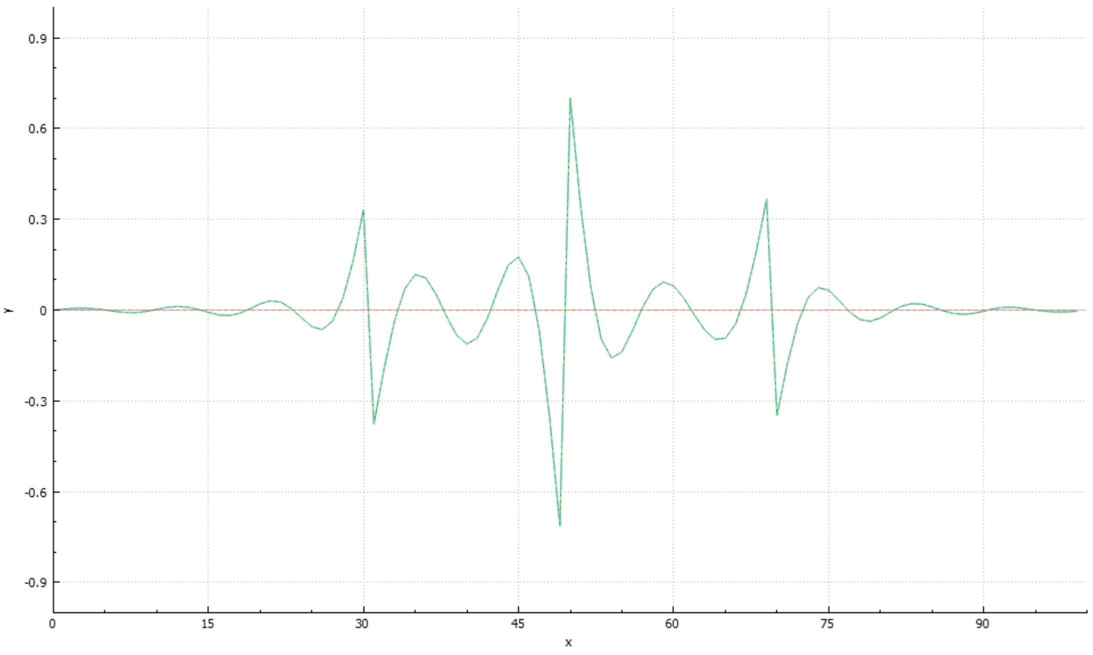


(b) Thousandth iteration, doubled standard deviation for the filter kernel.

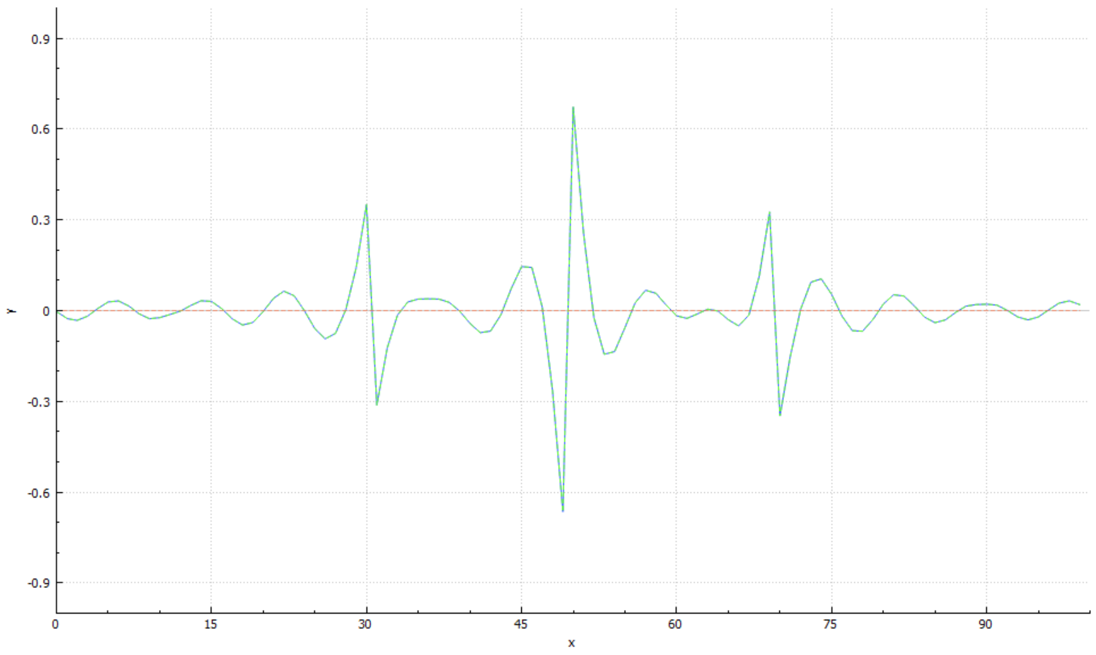
Figure 4.7.: High-pass filter artifacts on a step function with varying filter standard deviations.



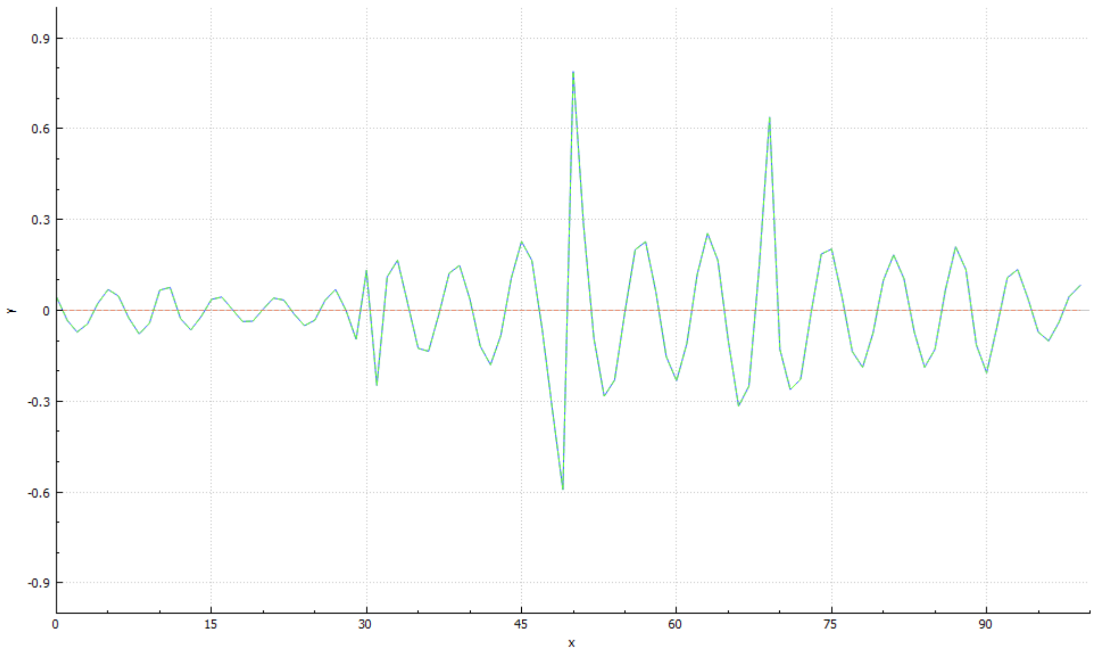
(a) First iteration.



(b) Fiftieth iteration.



(c) Thousandth iteration.



(d) Ten thousandth iteration.

Figure 4.8.: High-pass windowed sinc on a simple step function.

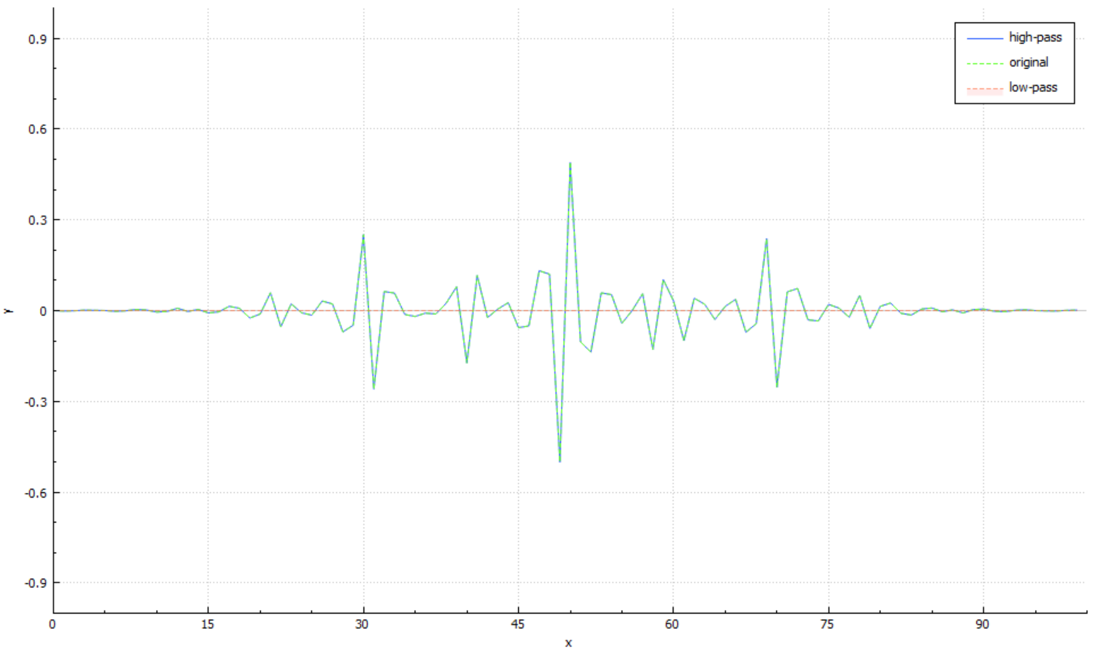


Figure 4.9.: High-pass Gaussian on a simple step function. Thousandth iteration, kernel width increased from 15 to 17.

It might seem paradoxical at first sight that the windowed sinc had slower artifact appearance, as the Gaussian should have the better spatial response. Of course, the filter is different by nature. However, it is windowed, because the sinc function never drops to zero in either direction. Interestingly, the Gaussian is also infinitely wide. This leads to the hypothesis the numerical instability might be related. Testing showed that using a wider kernel without changing the standard deviation has the effect of significantly reducing the artifacts (see figure 4.9). Increasing the width more reduces some of the artifactual peaks, but some remain regardless. This makes sense, given the information we found in [Luc14].

Cause

To summarize, the cause for the numerical instability lies, at least for the largest part, in truncation of the filter kernel. To explain what we believe to be happening, look at the frequency response of an ideal filter and a truncated sinc filter in figure 4.10. Even though we use a windowed sinc, due to kernel discretization there is likely still some implicit truncation, a small tail beyond the kernel's resolution. The frequency response for the truncated (high-pass) Gaussian that caused the artifacts will have similar issues. One or more frequencies in the response for this filter kernel are going slightly above the amplitude response of 1. Very slightly, because, as shown, the amplification the filter applies only becomes apparent after a very large number of iterations.

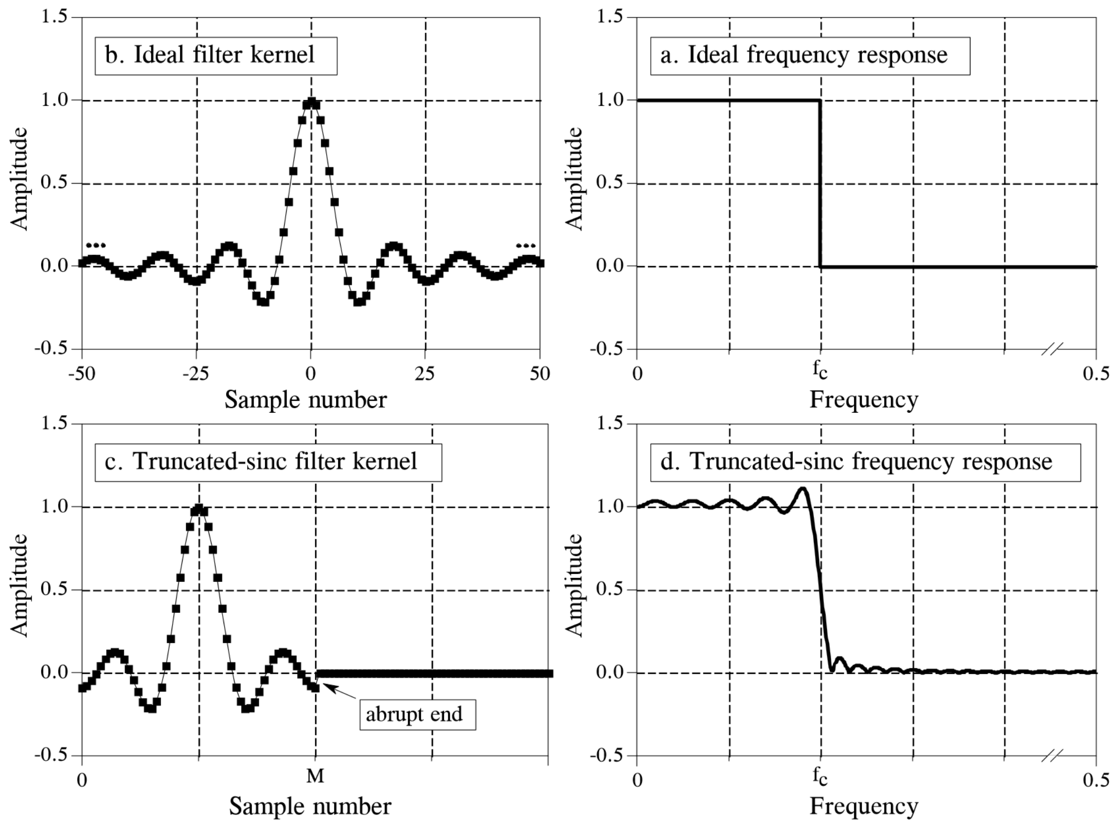


Figure 4.10.: Filters and frequency responses given a cut-off frequency F_c. Images taken from [S$^+$97].

Solution

There are two immediate solutions available: increase the kernel width or use a window function. Increasing the kernel width is simple, but comes at a slightly larger computational performance cost. Using a window function unnecessarily complicates things with regard to the frequency response.

A different solution to this problem was found in the self-damping of the shallow water simulation. By giving the water simulation more time to dampen itself, these artifacts do not become a problem. This solution is discussed in section 4.4.1. It is about linearly interpolating the current and envisioned water states. This solution makes the Gaussian a viable filter again, while saving some execution time.

The difference between a kernel width of 13 and a width of 15 is largely that a width of 13 causes the data to show instabilities approximately twice as fast. The first 50 iterations are very similar, save for some small extra artifactual peaks, so it might even be possible to make the kernel *smaller* and thereby gaining filter execution speed. We will leave it to future work to decide whether this is a good idea or not.

4.4. Water Control

This section describes our method to control the water simulation. We start with explaining the basic algorithm and an adjustment to parametrize strictness of the simulation control in section 4.4.1. After that, we will discuss the grid layout and how to interpolate the large-scale data in sections 4.4.2 and 4.4.3. We will then define the area in which we apply control in section 4.4.4, describe the details for the Gaussian filter mask in section 4.4.5 and finish with how to apply control on different parts of the simulation in section 4.4.6.

4.4.1. Algorithm

Every frame, the low-frequency components of the water state are separated from the high-frequency components using the Gaussian filter from section 4.2. The split roughly looks like presented in figure 4.11, where G is the low-frequency part after one application of the filter and I is the identity matrix. The low-pass filter is also applied on the interpolated control data. The high-frequency part of the water state is then added to the low-frequency part of the control data.

Using matrix-notation for converting from and to the frequency domain (for example with a DFT [S+97] matrix) and a matrix representing the Gaussian filter in the frequency domain, the method can be written down like this:

$$S_t = F^{-1}GFC_t + S_{t-\Delta t} - F^{-1}GFC_{t-\Delta t} \tag{4.21}$$

S_t and $S_{t-\Delta t}$ are the current and previous water states respectively. C_t and $C_{t-\Delta t}$ are the current and previous control data states. F is the matrix that maps to the frequency

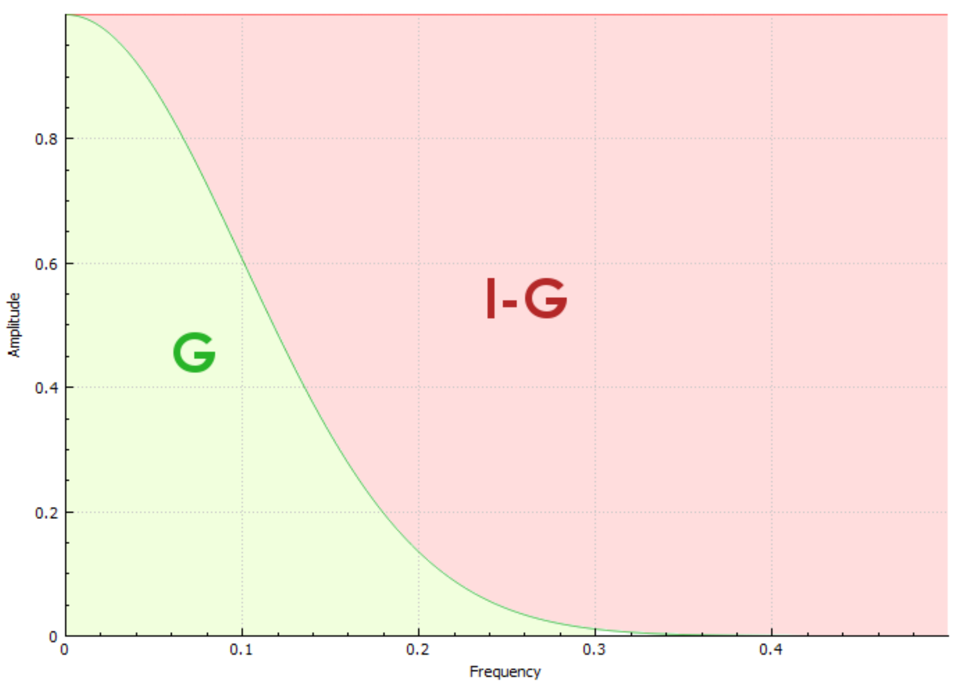


Figure 4.11.: Frequency domain split through Gaussian filter.

domain and F^{-1} its inverse. G is the low-pass Gaussian filter in the frequency domain, represented by a diagonal matrix.

Rewriting the formula gives:

$$\begin{aligned} FS_t &= GFC_t + FS_{t-\Delta t} - GFS_{t-\Delta t} && (4.22a) \\ &= GFC_t + IFS_{t-\Delta t} - GFS_{t-\Delta t} && (4.22b) \\ &= GFC_t + (I - G)FS_{t-\Delta t} && (4.22c) \end{aligned}$$

$(I - G)$ is the high-pass filter. Since the high-pass filter is applied on the simulation state each frame, any low frequencies in this data will quickly disappear. This includes the low-frequency information introduced by the control data. The highest frequencies remain longest, but due to the implicit damping of the water simulation these will not remain indefinitely.

Strict Versus Subtle Control

The control applied in the algorithm so far is still rather strict. Because the Gaussian high-pass filter filters practically every frequency to some extent, most signals disappear quickly. Additionally, along the large-scale free-surface border some minor oscillations can occur, due to the weakly controlled water simulation reacting on the corrections applied by the algorithm, which in turn corrects the reaction. In order to get rid of

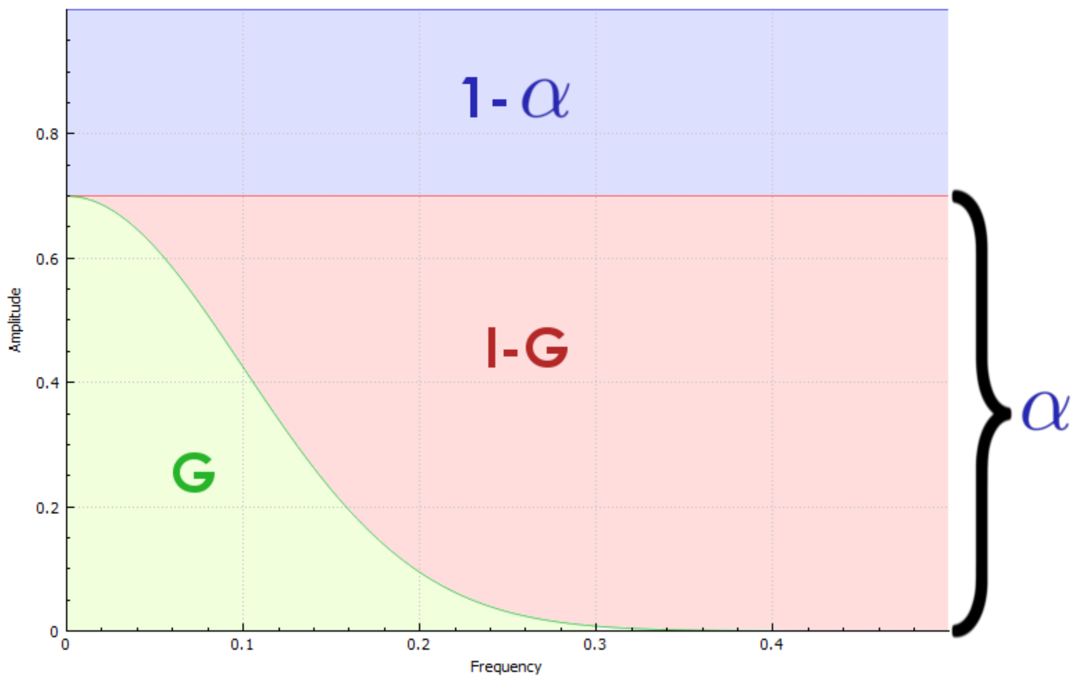


Figure 4.12.: Frequency domain split through Gaussian filter, blended with the original signal.

these problems, the state of the water, $S_{t-\Delta t}$, is linearly interpolated with the target state calculated by equation (4.21). We will call the user-defined interpolation factor α, set to 0.05 by default. This modifies our algorithm to become:

$$S_t = (1-\alpha)S_{t-\Delta t} + \alpha(F^{-1}GFC_t + S_{t-\Delta t} - F^{-1}GFC_{t-\Delta t}) \tag{4.23a}$$

$$FS_t = F(1-\alpha)S_{t-\Delta t} + F\alpha(F^{-1}GFC_t + S_{t-\Delta t} - F^{-1}GFC_{t-\Delta t}) \tag{4.23b}$$

$$= (1-\alpha)FS_{t-\Delta t} + \alpha(GFC_t + FS_{t-\Delta t} - GFS_{t-\Delta t}) \tag{4.23c}$$

$$= (1-\alpha)FS_{t-\Delta t} + \alpha(GFC_t + (I-G)FS_{t-\Delta t}) \tag{4.23d}$$

This shows that we also linearly interpolate between the frequencies of the previous and target state when we linearly interpolate between their signals. Put another way, we reduce the strength of the Gaussian filter. See figure 4.12 for a visual representation.

4.4.2. Grid layout

The raw control data used to test our model consists of an 18 by 18 cells grid of data, where each cell is 50 meters in width[2]. This data is a result of a water simulation on a

[2]The flooding framework (chapter 3) performs simulation on a quadtree. The leaves of this quadtree are between 50 and 200 meters in width. Because of this, we chose 50 meters for our test control data.

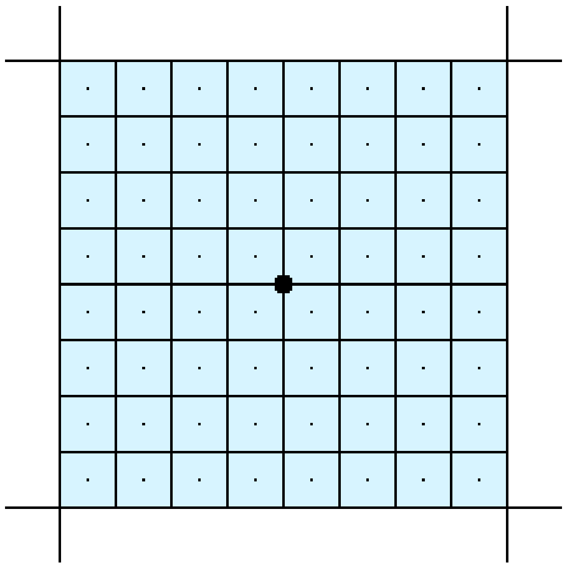


Figure 4.13.: Small and large scale grid layout.

grid of 18 by 18. The borders lie outside of the simulation domain and should not be used. Thus, the total domain on which the small-scale simulation is controlled is 16 by 16 large cells, or 800 by 800 meters. We use *ControlScale* = 8, which means each small cell is $50/8 = 6.25m$. The size of our small-scale grid is thus 128 by 128, which is 130 by 130 including borders. The cells are laid over each other so that the area they cover (excluding borders) neatly aligns, as in figure 4.13. The dots are the cell centers, the velocities are stored on the boundaries as described in figure 4.1b. This goes for both the large cell and the small cells.

Grid transformations

The position on a grid with respect to the scale can be described as follows. Say a position $\mathbf{x} = (x, y)^\intercal = (0, 0)^\intercal$ is in the bottom-left corner of the grid, including border cells. This is on the bottom-left corner of the bottom-left cell. A position of $(1, 0)^\intercal$ would then indicate the bottom-left corner of that cell's neighbour, and so on. We can use this conceptual sub-space to intuitively convert small-scale grid coordinates to large-scale grid coordinates and vice-versa.

Transforming from small- to large-scale coordinates is as follows. For water depth and height (h and H):

$$x_{large} = 0.5 + (x_{small} - 0.5)/controlScale \tag{4.24a}$$

$$y_{large} = 0.5 + (y_{small} - 0.5)/controlScale \tag{4.24b}$$

For x-velocity (u):

$$x_{large} = 1 + (x_{small} - 1)/controlScale \tag{4.25a}$$
$$y_{large} = 0.5 + (y_{small} - 0.5)/controlScale \tag{4.25b}$$

For y-velocity (v):

$$x_{large} = 0.5 + (x_{small} - 0.5)/controlScale \tag{4.26a}$$
$$y_{large} = 1 + (y_{small} - 1)/controlScale \tag{4.26b}$$

Transforming from large- to small-scale coordinates should not be needed, but otherwise requires only a simple inversion of the above equations.

4.4.3. Large-scale data interpolation

In our test scenarios, the large-scale data is simply a water simulation executed on the same terrain as the small-scale simulation, but with larger cells. This data needs to be scaled to match the small-scale simulation it can be used. Spatially, this is done by projecting the data on a higher resolution grid and bilinearly interpolating data points that lie in between. High-frequency details of this data, most notably the edges between data points, become visible in the simulation. To that end, this interpolated state is filtered with the same low-pass Gaussian filter as the one used for dividing the low and high frequencies. We initialize the small-scale simulation with the first spatially interpolated frame of this large-scale data.

The large-scale data used might not be suited for bilinear interpolation, it could for example be stored in a quad-tree. If this is the case, it will have to either be subdivided first or be interpolated with a different interpolation scheme.

Since subsequent spatial interpolations may be far apart in time, we then interpolate each cell separately over time to fill in the missing data. Since our data is uniformly spaced, we use cubic b-spline interpolation. An algorithm for this can easily be found on the web. We believe this to result in a more natural transition between different time frames than linear interpolation. In order to be able to do this, we need to store 4 frames of spatially interpolated data at all times: the closest frame whose time we have passed and the 3 frames following it. Another interesting method could for example be cubic Hermite spline interpolation, in which the interpolated data will always move through its data points, but this is more computationally expensive.

Surface interpolation

Surface interpolation is done by interpolating the water height, η. However, the bottom elevation used for the small-scale simulation is real data and not a linear interpolation. As such, simple linear interpolation can create water where it should not be. See figure 4.14. The blue dots indicate the large-scale water height, where the right cell is dry. The blue line is the interpolation and brown is the small-scale bottom elevation. The erroneous water is marked with red. To counteract this effect, we use a weighted linear

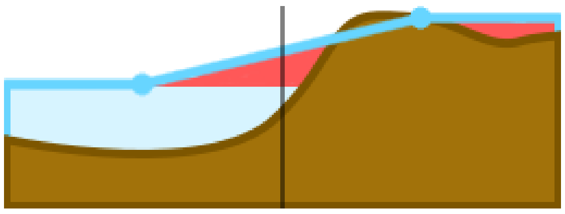


Figure 4.14.: Surface interpolation errors.

interpolation. Note that time interpolation can cause similar appearances, for example when future water spreads out over terrain that lies higher than the current water level.

Suppose we are interpolating a grid variable s at a position $(x, y)^\intercal$. We have determined the four corners between which our position lies to be s_{00}, s_{10}, s_{11} and s_{01}, starting bottom-left, going counter-clockwise. If we convert our position to the coordinate system where those points lie on $(0,0)^\intercal$, $(1,0)^\intercal$, $(1,1)^\intercal$ and $(0,1)^\intercal$ respectively, $(x', y')^\intercal$, linear interpolation is as follows:

$$s(x, y) = w_{00}s_{00} + w_{10}s_{10} + w_{01}s_{01} + w_{11}s_{11} \tag{4.27}$$

The weights w are:

$$w_{00} = (1 - x')(1 - y') \tag{4.28a}$$
$$w_{10} = x'(1 - y') \tag{4.28b}$$
$$w_{01} = (1 - x')y' \tag{4.28c}$$
$$w_{11} = x'y' \tag{4.28d}$$

To prevent described interpolation errors a multiplier is applied to each weight, c_{xy}, where $_{xy}$ denotes the corresponding corner. c_{xy} is 1 if the corresponding cell is wet or if the cell is dry but below the highest water level between the 4 cells. Otherwise, c_{xy} is 0. Note that this means we need to apply equation (4.10) on the large-scale data. If there is no depth data for the large-scale simulation, the bottom elevation of the small-scale simulation can be used to calculate it.

$$c_{xy} = \begin{cases} 1 & \text{if } s_{xy} \text{ is wet or } s_{xy} < max(\{\text{all wet } s_{xy}\}) \\ 0 & \text{otherwise} \end{cases} \tag{4.29}$$

After that, the final value is normalized by dividing by the accumulated weight of all cells. Of course, this only works if there is any weight at all. Additionally, the result must not end up below the bottom elevation of the small-scale simulation, H. This modifies our interpolation method to become:

$$s(x, y) = \begin{cases} \max(H(x, y), \frac{\sum(cws)_{xy}}{\sum(cw)_{xy}}) & \text{if } \sum(cw)_{xy} > 0 \\ H(x, y) & \text{otherwise} \end{cases} \tag{4.30}$$

It is important to clamp to H(x,y), which can otherwise cause oscillations close to dry cells in controlled regions.

Velocity interpolation

Because reflective faces at domain and dry/wet borders have their velocity set to 0, interpolating these values will incorrectly drag the 'correct' velocities towards zero. A moving front of water does not have a velocity of zero. Because of that, the velocity interpolation is similar to the surface interpolation. Instead of checking if the cell is dry or wet, we have to check if the face is reflective or not with equation (4.11). Note that our slight modification of that equation is relevant here.

$$c_{xy} = \begin{cases} 1 & \text{if } s_{xy} \text{ is } \textit{not} \text{ reflective} \\ 0 & \text{otherwise} \end{cases} \tag{4.31}$$

$$s(x, y) = \begin{cases} \frac{\sum(cws)_{xy}}{\sum(cw)_{xy}} & \text{if } \sum(cw)_{xy} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.32}$$

4.4.4. Control region

Control is not applied everywhere. This is to make sure the free-surface water border can behave naturally. This control region is decided based on our large-scale data.

Every cell of large-scale data $S_L(i, j)$ is assigned a value, $c_L(i, j)$, called the *control contribution*. This value is assigned -1 for dry cells, 0 for wet cells that share an edge with one or more dry cells and 1 for wet cells that only share edges with other wet cells. This large-scale grid of values is then projected onto a small-scale grid and linearly interpolated. This gives us a grid of values which can be used to determine where control needs to be applied. We will refer to this value as $c_{i,j}$. It is updated together with interpolating the large-scale data.

4.4.5. Gaussian filter masking

Since we're only interested in the water, we do not want to include dry areas when applying the Gaussian filter. They would introduce a lot of invalid data into the system. These are excluded by masking them from the filter as described in section 4.2.2. Uncontrolled areas are masked for a similar reason, we need the high-frequency details only for the controlled region. This mask is applied to every filter operation described in this thesis. The mask needs to be updated every frame after large-scale data interpolation, but before the Gaussian filter is applied on the interpolated state.

Since the water state $S(x, y)$ is stored in a staggered grid format, separate masks are used for the water surface and each of the velocities. For the water surface, the domain boundary cells, all dry cells and all uncontrolled cells are masked. For the velocities, all faces that border the domain boundary, all reflecting faces and all uncontrolled faces are masked.

The decision to mask uncontrolled faces is also based on experimenting. Not masking uncontrolled regions introduced artifacts that clearly visualize the control region boundaries. However, these experiments were solely based on strict control (section 4.4.1), so more experimenting might be a good thing.

4.4.6. Applying control

Note that whenever control is applied and η or h is updated, h or η respectively will have to be updated with it. Also, the domain boundaries should be masked in any filter operation and excluded from control, but do have to be updated afterwards as in section 4.1.3.

We apply control on cells where $c_{i,j} > c_b$. Here c_b is the control boundary threshold, 0 by default. When varying this value, it is very important to make sure that the outer border of the control region consists solely of cell centers and not edges. The easiest way to achieve this is to use different thresholds for the water surface and the velocities, where the threshold for the velocities is at least $1/\mathit{controlScale}$ smaller. If the outer border consists of cell edges, where the velocities are stored, it will disrupt interaction between the controlled region and the free region.

The control threshold, c_{ct}, by default 1, is the value from which point onwards control is applied as discussed in section 4.4.1. A possible exception to this are dry cells and cells that have one or more dry neighbours. This way, controlled water can be pushed away and will naturally fill up again. A (large) downside to this is that water can not be created on dry cells if the large-scale data requires it, for example if the small-scale bottom elevation prevents water from spreading to a certain region. We have no good solution for this, so we currently decide to do this or not based on what the situation requires.

The area where $c_b < c_{i,j} < c_{ct}$ we call the control boundary region. In this region, control is applied just as in section 4.4.1, but the linear interpolation factor α is multiplied with a value based on how deep in the control region it is:

$$\alpha \mathrel{*}= \frac{\min(c_{i,j}, c_{ct}) - c_b}{c_{ct} - c_b} \tag{4.33}$$

This way, control will slowly decrease in strength towards the outside of the boundary.

It may be a good idea to not apply control on the velocities at all in the boundary region. It may also be a good idea to more loosely control the velocities in general, or even not at all, as they are less important for the visual quality than surface height. Again, this depends on the situation and the purpose of controlling the simulation.

In addition to controlling the water, water is slowly removed from regions where $c_{i,j} = -1$. The water depth h is multiplied with 0.999 every frame. This way, if some water is leaked, it will be cleaned up eventually. In some cases this might not be good enough.

A good alternative to this straightforward water removal is to apply the same kind of control as discussed on removal regions, but with a smaller interpolation factor and only on the water height with a target state that is equal to the bottom elevation. This would prioritize removal of the large mass of water, but leave more of the details intact.

5. Results

This section is about analysis on the method and its parameters. We will start with describing some qualitative criteria for our simulation control in section 5.1 and describe the scenarios used to test these criteria in section 5.2. In section 5.3 we will discuss the parameters that are varied through several of those scenarios. We will then describe both quality and performance of our test results in sections 5.4 and 5.5. We conclude with an evaluation in section 5.6.

5.1. Qualitative criteria

There are two main goals that our simulation ideally achieves: controlled behaviour that matches with our large-scale data and realistic water behaviour. Truly realistic behaviour and perfectly controlled behaviour do not match. They contradict; truly realistic water does not necessarily flow how we want it to, which is the whole reason for controlling it. However, there are visual characteristics of water movement that we can maintain, making our simulation look like water while still steering it in the right direction.

We had previously determined that the shallow water simulation is a good trade-off between realism and performance when it comes to visualizing water-land interactions. Assuming this model is good for our purposes, we need to maintain the visually important characteristics of this model as good as we can. We have tried to achieve this through filtering the high frequencies in the water state from the low frequencies and subsequently replacing the low frequencies. This way, we hope to keep as much of the visually important details as we can, while still having control over the large-scale flow of the simulation. To maintain the details longer, we introduced linear interpolation with a target state, literally keeping more of the original simulation, skewed towards the high-frequency details.

The visual behaviour of the controlled water simulation is rather uninteresting. To test how the water reacts on high-frequency details, we have created a controlled simulation where we add water drops over time. The waves introduced by these drops should look like water, but the large-scale simulation should be relatively unaffected. We will use our own eyes and opinion to judge the quality of the simulation's reaction.

In addition to good reaction to high-frequency details, we believe the controlled simulation should look like an uncontrolled simulation that comes from a similar origin. By this we mean the forces that act on the water state are similar. We use an uncontrolled large-scale simulation for controlling the small-scale simulation. Similar to this, we have created an uncontrolled small-scale simulation that adds the same amount of water per second in the same region as this large-scale simulation. Determining the quality of

Figure 5.1.: Heightmap of our simulation domain.

our simulation is then a matter of visual resemblance between the controlled and uncontrolled small-scale simulations. We use a mathematical approach to determine this resemblance, discussed later in section 5.4.

5.2. Scenarios

The following is a description of the scenarios used to test the control method. Each of these scenarios takes place on the same domain and will be run for 10 minutes unless mentioned otherwise.

Large-scale simulation

The large-scale data we use to control the small-scale simulation is the result of a large-scale simulation on the same domain, covered with 16 by 16 cells. The layout has been described in detail in section 4.4.2. Our custom flooding scenario takes place on an area defined by the heightmap in figure 5.1. The width in x- and y-direction is 800 meters and the height ranges from 0 to 50 meters, given by $heightmap_{x,y} \cdot 50$. Every frame, we add $0.15 \cdot \Delta t$ to the water height in 4 neighbouring cells, as highlighted in figure 5.2.

Controlled small-scale simulation

The controlled small-scale simulation cells are 8 times smaller in each dimension. They have been laid out relative to the large-scale grid as described in section 4.4.2. We apply control on it as described in section 4.4. This scenario is executed multiple times with different control parameters, which will be described in section 5.3.

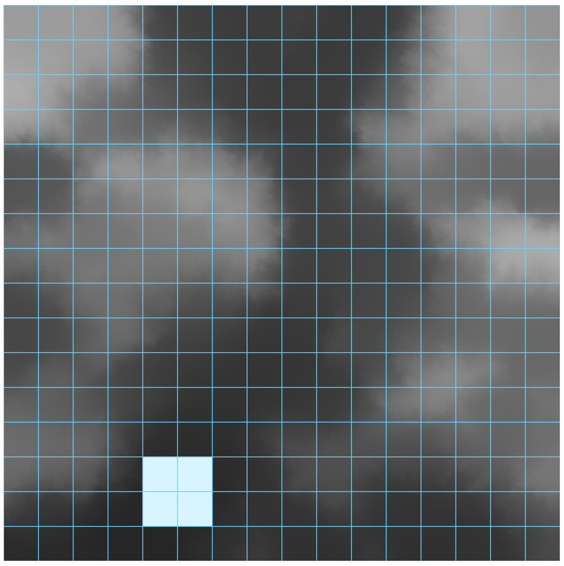


Figure 5.2.: Large-scale cells where water is added each frame are highlighted.

Uncontrolled small-scale simulation

The uncontrolled small-scale simulation follows the layout of the controlled simulation. There is no control applied. Instead, in every frame, we add $0.15 \cdot \Delta t$ to the water height in the same area as in figure 5.2, which is covered by 16 by 16 small cells.

Water drops on controlled simulation

This scenario is similar to the controlled small-scale simulation. We use the default parameters for this, which will be described later. We will create videos for a part of the simulation.

In addition to applying control on the simulation, we drop large amounts of water on the simulation over time, with varying intervals and positions. These drops are just masses of water added to the simulation. This should give a good impression of the robustness of the controlled simulation. In addition, we will have clear visual feedback of how the high-frequency filter performs in practice.

5.3. Parameters

In this section, we will discuss the different parameters that will be varied in the controlled small-scale simulations. While there are a lot of parameters that can be tweaked, most of them have very little influence on the result or only deal with border regions. Adding in practical reasons, we have decided to only vary the two most important parameters that work on controlled regions.

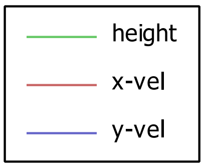


Figure 5.3.: Legend for the graphs in section 5.4.

5.3.1. Filter cut-off frequency

The cut-off frequency defined in the frequency domain, f_c, is inversely proportional to the filter's standard deviation in the spatial domain and thus the kernel width. The relation was described in section 4.2.4. A lower cut-off frequency means less strict control, most notably with regard to the frequencies higher than the cut-off frequency. Additionally, a wider kernel will result in slower performance.

The default value for this is $f_c = 1/(2 * \mathit{ControlScale})$ (see equation (4.18)), where $\mathit{ControlScale} = 8$. The values we test with are 2, 1, 0.5 and 0.25 times the default: 0.125, 0.0625, 0.03125 and 0.015625.

5.3.2. Water state control strength

α, the linear interpolation factor for interpolating the current simulation state with the target state. A lower interpolation factor means control is less strict. The result of this is that the mass and velocities might lag behind on the large-scale simulation. However, loose control also means there is more room for free water movement, with an emphasis on the higher frequencies.

The default for this interpolation value is 0.05. In addition to the default value, we will also test 0.5, 0.1, 0.02 and 0.01.

5.4. Quality comparison

This section tries to determine the quality of the simulation control. We will try to determine the independent quality of the simulation as a whole, as well as the quality of the different simulations relative to each other. Every part of the water state, η (surface height), u (x-velocity) and v (y-velocity), will be graphed separately. The colors for each of them are shown in figure 5.3.

5.4.1. Controlled vs. uncontrolled

This subsection is about comparing the controlled simulations to the uncontrolled simulation, our "ground truth". To first get a grasp of what the simulation looks like, take a look at our video located at `https://youtu.be/ybihrgH0EnI`. It shows the relative error of the water surface height (as defined later in equation (5.1c)) between a default controlled simulation and an uncontrolled simulation that mirrors the large-scale input forces. Black means there is no water in either simulation. White means the relative error is 0. Dark red and dark blue mean the relative error is -1 and 1 respectively. In words, this means that dark red implies there is water in the uncontrolled simulation, but not in the controlled simulation. The other way around for blue. The video is sped up by a factor 10.

From the video, a few distinct phases in the uncontrolled simulation can be identified, with approximate time in seconds:

0 seconds: Initial rest state. The simulations are the same.

0 to 120 seconds: Water is added from frame 1 onwards, creating waves that disperse outwards and reflect on the boundaries. This creates a large wave in the uncontrolled simulation, which shows up as an error in the video. The uncontrolled simulation is ahead of the controlled simulation in the top region of the water. Some high-frequency errors show up in the bottom-right area of the simulation.

120 to 170 seconds: The bottom lake fills up. The uncontrolled simulation is still ahead of the controlled simulation.

170 to 240 seconds: A small appendage above the bottom lake fills up too. The uncontrolled simulation is still ahead.

240 to 340 seconds: The high-frequency errors in the bottom-right dissolve. The lake reaches a threshold surface height, water starts flowing over an edge and streams in to the top lake. The controlled and uncontrolled simulations show large differences in the top region of the simulation.

340 to 600 seconds: The top lake fills up. High-frequency errors in the top region remain until the end of the simulation, but gradually become less pronounced.

Overall, the simulations show great similarity on a large scale. The highest errors are in the border regions, where there is a lot of water flow. Most of the areas with large errors are combinations of red and blue, meaning they are high-frequency errors.

RMSE and modifications

In order to compare the simulations, we use the root mean square error (RMSE) and two modifications of the RMSE. We store the results of these functions for every frame and display them in graphs. The functions used are shown in equation (5.1), where x' is a state variable of the uncontrolled simulation and x is a state variable of the controlled simulation. Note that we compare all indices excluding borders, but only divide by the number of grid values that are valid in the ground truth. This ensures the result is only relative to the ground truth. Invalid grid values for the water surface η are those that are dry. For the velocities u and v, all reflecting edges are invalid (which includes edges between dry cells).

The RMSE, equation (5.1a), is the standard deviation of the error. The result is in the same units as the variable we are comparing. This is meters for the surface height, η, and meters per second for the x- and y-velocities, u and v.

The first modification of the RMSE we call the normalized RMSE or NRMSE, equation (5.1b). We normalize the RMSE with the range of the values in the uncontrolled simulation. The result can be interpreted as a percentage-based error, where 1 equals 100%.

The last modification is the root mean square of the relative error, RMSRE, equation (5.1c). We only use this for the water surface height. We define the relative error as the difference between two values divided by the values added. The result of this is that the error is maximized when there is some water present in one simulation, but none in the other. This way we can objectively verify that the controlled and uncontrolled simulations are similar in shape. The result of this can also be interpreted as a percentage-based error, however we only used this to show the video, as the result is largely dependent on water depth and as such is prone to misinterpretation. This is especially true for our case, as the borders are uncontrolled and depth is lowest here.

The results of the RMSE and its modifications represent accuracy, in a numerical sense. A low RMSE does not necessarily mean visual accuracy, as shape and structure are not variables.

$$RMSE(x_t, x'_t) = \sqrt{\frac{\sum_i \sum_j (x_{i,j,t} - x'_{i,j,t})^2}{numValid(x'_t)}} \tag{5.1a}$$

$$NRMSE(x_t, x'_t) = \frac{RMSE(x_t, x'_t)}{\max(x'_t) - \min(x'_t)} \tag{5.1b}$$

$$RMSRE(x_t, x'_t) = \sqrt{\frac{\sum_i \sum_j (\frac{x_{i,j,t} - x'_{i,j,t}}{x_{i,j,t} + x'_{i,j,t}})^2}{numValid(x'_t)}} \tag{5.1c}$$

RMSE analysis

First, let us take a look at the RMSE originating from the default control parameters, figure 5.4. Ignoring the initial state, the surface height RMSE ranges from ≈ 0.13 to $0.3m$. The y-velocity varies from 0.32 to $0.73ms^{-1}$ and the x-velocity varies more, from 0.45 to $1.01ms^{-1}$.

This section will contain screenshots that show the square error of the simulation at varying points in time. This is to give an idea of what the simulation looks like at that point in time.

The surface height error is highest in the first 20 seconds. See figure 5.5, which shows the square error at 10 seconds. This screenshot is from the square error video, ., which seems to be caused by the difference in water being added. The controlled simulation interpolates with an interpolated and filtered water height target, which is more spread out than the direct addition of water mass as done in the uncontrolled simulation. In the first minute, the velocities also show a few peaks, which are likely related to the same waves. From there on, until 180 seconds, the errors for surface height and y-velocity remain somewhat stable, while x-velocity peaks. This seems related to the movement

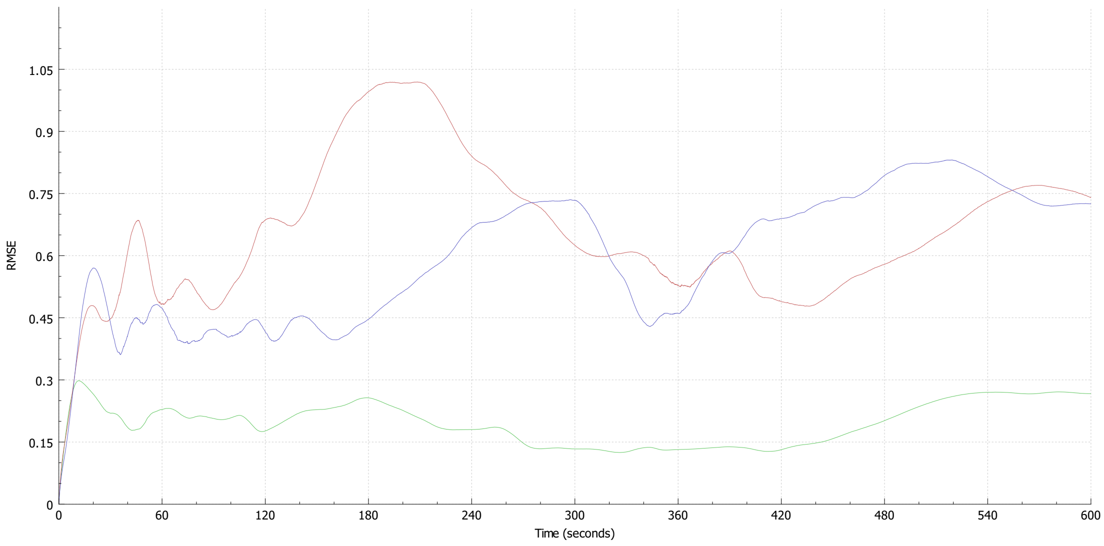


Figure 5.4.: RMSE, $f_c = 0.0625$, $\alpha = 0.05$ (default parameters).

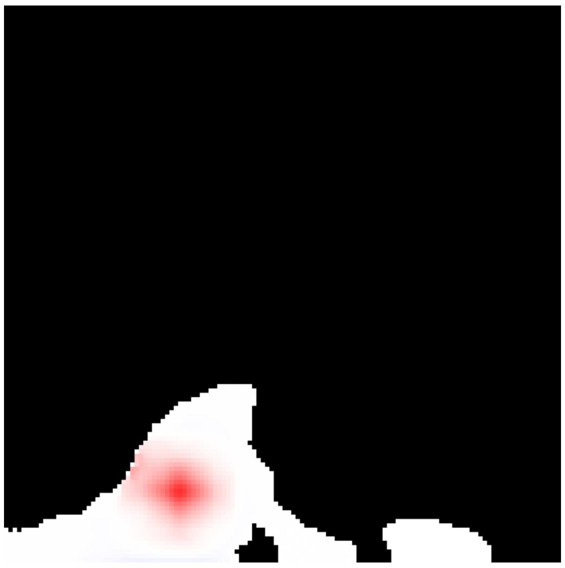
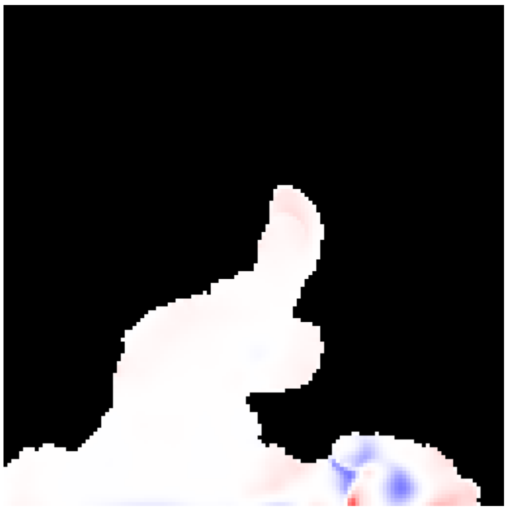
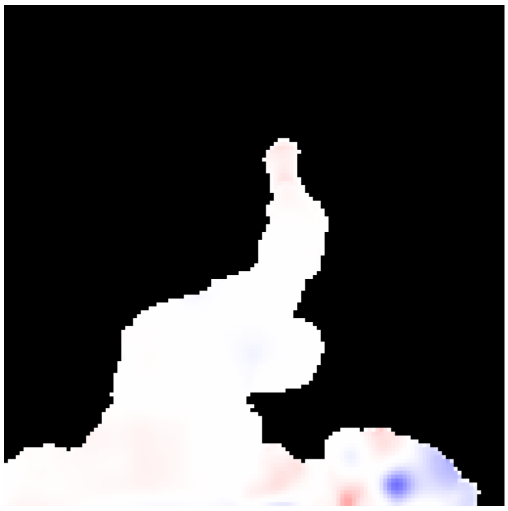


Figure 5.5.: Frame 200 (10 seconds) of the video showing the squared errors between a controlled simulation with default parameters and an uncontrolled simulation. Deep blue and red mean a high error, white means a low or no error, black is dry.

happening in the bottom-right corner of the domain, where the simulations seem to disagree quite strongly, as both videos show as well.

From ≈ 210 seconds (figure 5.6a), while the x-velocity error declines, the y-velocity error increases. The water is stabilizing slightly. This is just before the point in time


(a) Squared error video, 210 seconds.


(b) Squared error video, 240 seconds.

where the uncontrolled simulation starts streaming water into the top lake. It has been ahead of the controlled simulation for a while now, which was causing the y-velocity error to increase (since it is flowing upwards).

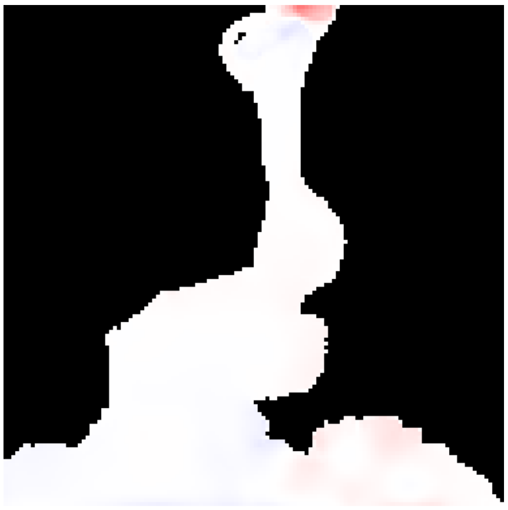
After that, at 240 seconds (figure 5.6b), the water flows in to the top lake. Because the error is small, since there is little mass, the surface height error does not reflect the relatively large error in shape and instead declines, because the large bottom lake threshold is equal and thus the overall surface height is very similar in both simulations.

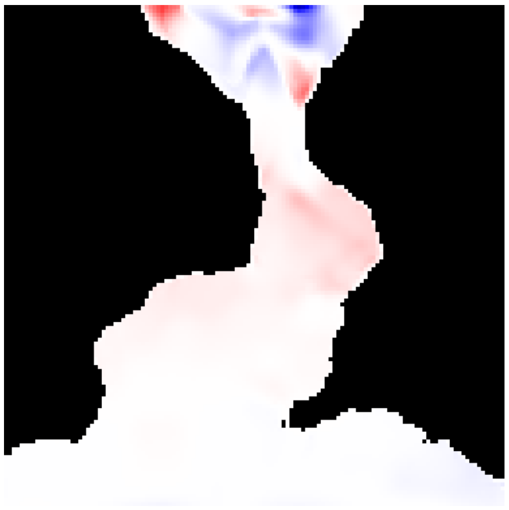
At around 340 seconds (figure 5.7a), the y-velocity error reaches a low peak. This is the moment where the water streaming into the top lake reaches the top domain boundary. The simulations are very similar at this point in time. After that, the top lake starts filling up. The simulations flow water in at different speeds here, with the uncontrolled simulation being slower, which causes a lot of relatively small-scale errors, as can be seen in figure 5.7b. It also causes a relatively large error in the small appendage between the two lakes.

RMSE comparison

The different parameter values we have tested create rather similar graphs. A higher α values creates larger errors for the velocities, while the error in mass stays approximately the same. The latter is probably due to the free boundaries and the limited number of directions the water can flow in. The higher errors in velocity for stronger control are go unexplained. Fact is that every graph shows weaker control gives a significantly lower velocity error. Loose control has a mixed effect on the surface height error. Weaker control causes a higher error in the start and during the time where water starts flowing into the top lake, but otherwise reduces the error.

The different cut-off frequencies have a smaller effect on the graphs. In combination


(a) Squared error video, 340 seconds.


(b) Squared error video, 500 seconds.

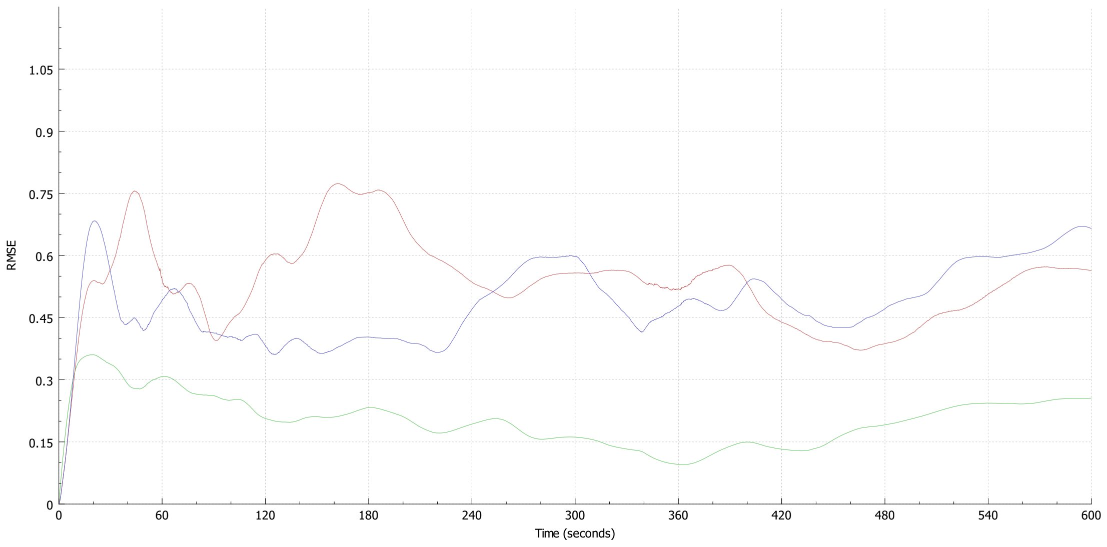


Figure 5.8.: RMSE, $f_c = 0.015625$, $\alpha = 0.01$ (best performing parameters).

with $\alpha \geq 0.05$, a cut-off frequency of 0.03125 (0.5 times the default) performs best. However, for more loose control, a cut-off frequency of 0.015625 (0.25 times the default) performs better. In fact, the best results for the velocities are obtained by the combination of $f_c = 0.015625$ and $\alpha = 0.01$. For the surface height, there is almost no difference between the different cut-off frequencies, with the exception of $f_c = 0.015625$, where the errors are slightly lower for loose control and higher for strong control. The graph of the best performing parameter combination is shown in figure 5.8. Overall, the surface height errors are larger in the start and during the overflow phase compared to all other simulations, but lower at the other intervals.

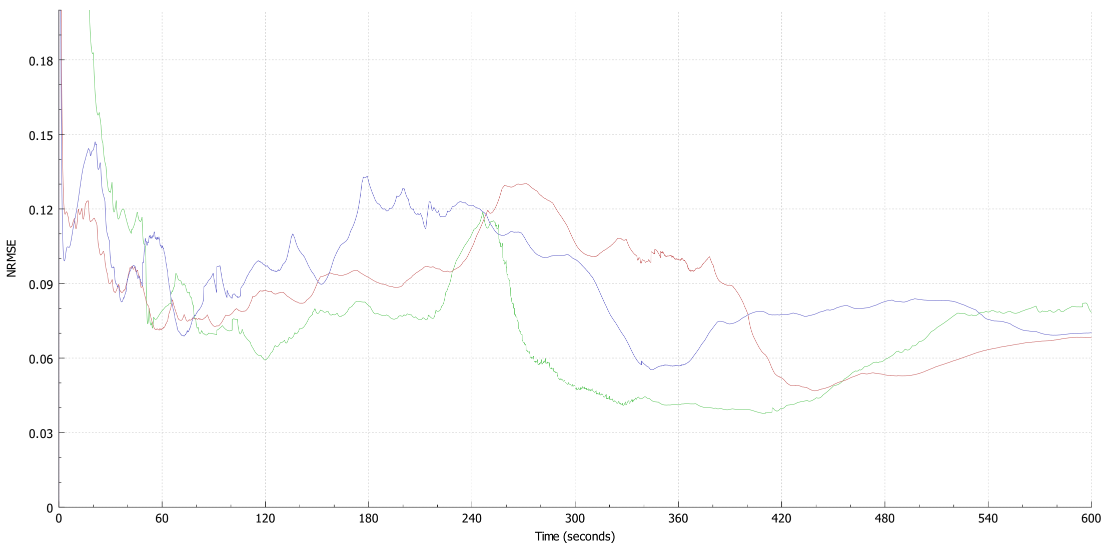


Figure 5.9.: NRMSE, $f_c = 0.0625$, $\alpha = 0.05$ (default parameters).

NRMSE analysis

Again, we start with a look at the default parameters, see figure 5.9. Initially, the errors are very large for the surface height. This is because the initial range of the surface height is very small. After the first minute, the error stays between 4 and 12%. The velocities both hang between 5 and 13%, with the x-velocity performing slightly better overall. This seems to be in agreement with the overall flow direction, which is in the y-direction.

The graph tells us somewhat the same story as the RMSE graph (figure 5.4) did. It also tells us the x-velocity error peak in the RMSE graph is maybe not as bad as it seemed, because apparently the range of values at this point in time is much wider as well.

There is one major exception: there is a peak in surface height error around 240 seconds. This peak is right at the time the water starts flowing in to the top lake. At this point in time, the range of surface height is shrinking while the water surface stabilizes. As a result, the normalized error grows dramatically with $\approx 5\%$ in half a minute, only to shrink even more afterwards. The shrinking is due to the surface height range increasing, because of the water flowing down.

At around 420 seconds the errors for surface height and x-velocity can be seen rising again. At this point in time, the range is decreasing again, while the total error is also slightly rising.

NRMSE comparison

The different graphs for the NRMSE also tell a story very similar to those of the RMSE. For the velocities, the same parameters come out as clearly the best. For the surface

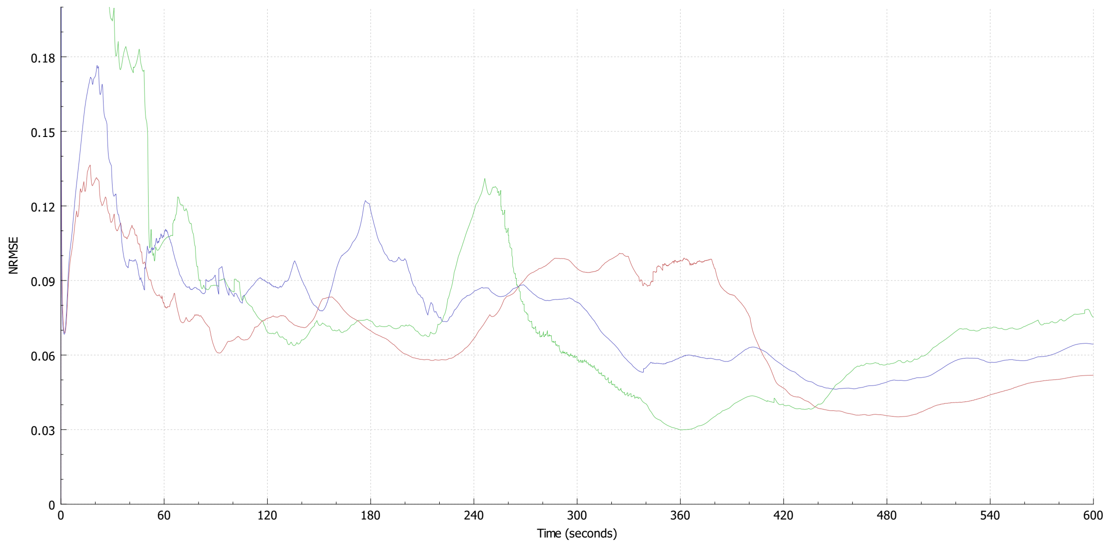


Figure 5.10.: NRMSE, $f_c = 0.015625$, $\alpha = 0.01$ (best performing parameters).

height, weaker control increases the peak, but lowers the rest of the errors. The difference between the different cut-off frequencies is marginal. Lower cut-off frequencies tend to favour weaker control, while higher cut-off frequencies tend to favour stronger control. The peak increases for weaker control, which we believe is due to the controlled simulation lagging slightly more behind on the uncontrolled simulation.

Squared error video analysis

Similar to the video showing the relative error, we also have a video showing the squared error for the water surface height: `https://www.youtube.com/watch?v=NBQMfudJUUk`. The largest squared error measured in this video was 3.05, which equals an error of 1.75. All coloring in the video is based on this maximum, with red indicating more water in the uncontrolled simulation and blue meaning more water in the controlled simulation. Unfortunately, there was not enough time to create videos for the x- and y-velocities as well.

In the video we can see many high errors occur close to the border regions. Most of the errors are strong variations over small distances, where positive and negative errors exist right next to each other, such as figure 5.6a. One major exception to this is the error in the appendage between the lakes, which is clearly a solid red (figure 5.11). There is also a small blue area in the bottom-right, slightly more difficult to see. The solid red area means the uncontrolled simulation has (relatively) significantly more water there. We see one possible cause: the large-scale water state is different from the uncontrolled small-scale water state. The error occurs in a controlled region and the default control does not apply control weak enough to allow for such a large deviation in such a large area.

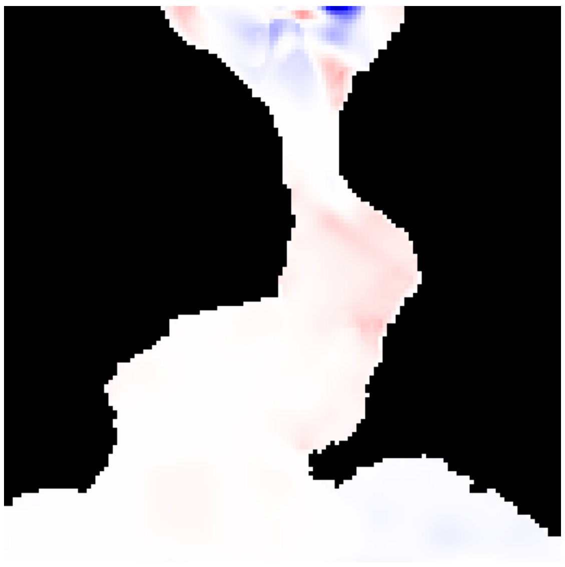


Figure 5.11.: Frame 9408 (470.4 seconds) of the video showing the squared errors between a controlled simulation with default parameters and an uncontrolled simulation. Deep blue and red mean a high error, white means a low or no error, black is dry. The largest squared error measured in this video was 3.05, which equals an error of 1.75.

5.4.2. Water drop analysis

This is an analysis of the previously mentioned water drop scenario. For this, it is best to look at our video using the default parameters: `https://youtu.be/sjn8Rr7XgYQ`. The corners of the green lines are the large-scale simulation cell centers, the blue surface is the controlled water. Visible are quite a few drops of water, most of which land outside our controlled water region. The water level of the controlled area (the center area of the lake) remains relatively stable when they do land on or close to the controlled region, large incoming waves are quickly reduced to smaller waves. These smaller waves remain for a while until they eventually disappear completely.

For reference, we also have a video that introduces water drops to water without control: `http://youtu.be/sZWdO2RfdF0`. The waves caused by the water drops in this video remain much longer. Unfortunately, the damping by the Gaussian and the interpolation has a strong effect on the existence of smaller waves later in time.

With more loose control and a lower cut-off frequency, things are already much better, as can be seen in another video, where the 'best' parameters from the previous analysis are used: `https://youtu.be/x041Dchb25w`. The waves in this video behave much like those in the uncontrolled simulation, dispersing almost as slow (the difference is hard to see with the naked eye).

We can conclude that the controlled area reacts well, when control is not too tight. However, the same can not be said for the areas that are supposed to be dry. They remain wet for a very long time. If they have to be dealt with stronger removal is required.

5.5. Performance

This section discusses the performance of our algorithm. Before we dive into it, some background information is useful.

The application implementing the algorithm is written on Windows in C++. It runs on a single Intel(R) Core(TM) i5 CPU @ 2.53 GHz in a default release x64 configuration of Visual Studio 2012. We have recorded the total time before and after the code implementing said parts of the algorithm. The execution time is given by the differences of these timings. The code used to retrieve the current time in milliseconds is given by listing 5.1.

```
double milliseconds_now()
{
  static LARGE_INTEGER s_frequency;
  static BOOL s_use_qpc = QueryPerformanceFrequency(&s_frequency);

  LARGE_INTEGER now;
  QueryPerformanceCounter(&now);
  return (1000 * static_cast<double>(now.QuadPart)) / s_frequency.QuadPart;
}
```

Listing 5.1: Code used to retrieve total milliseconds.

For every simulation we perform, the execution time of the following algorithm parts was stored:

- large-scale data interpolation;
- filter mask updating;
- Gaussian filtering;
- application of control;
- and shallow water solver.

Execution time analysis

The performance for a simulation with default parameters is shown by figure 5.12. There is an irregular pattern, different for every simulation, which is probably caused by other applications using the CPU. Ignoring that, what catches the eye right away is the increase in execution time as the simulation progresses. Looking closer, it becomes clear that at least a very large part of this, if not the whole, is caused by an increase in filter execution time. This makes sense, combining the facts that our controlled area increases over time and that only controlled cells are convoluted with the filter.

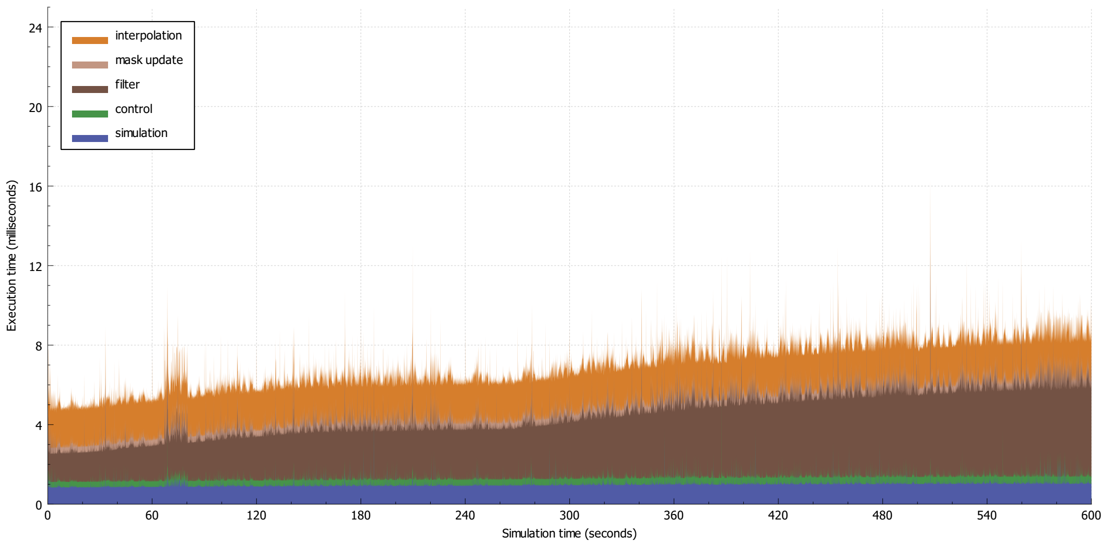


Figure 5.12.: Performance chart for default parameters, $f_c = 0.0625$, $\alpha = 0.05$.

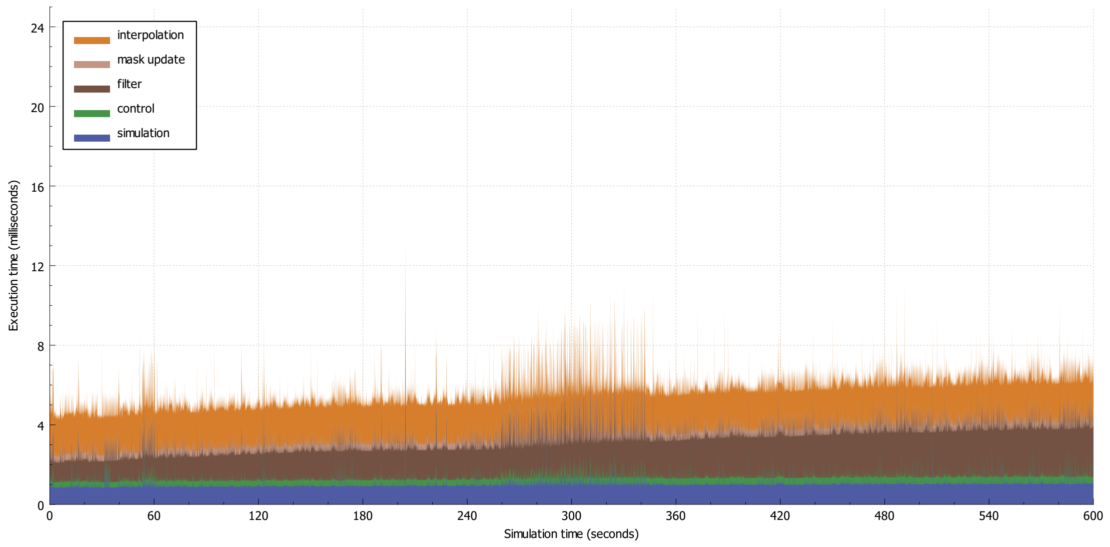
The execution time follows the simulation's progress; it increases, then stabilizes somewhat around 240 seconds, then increases again. Not so easy to see is that the execution time for the water simulation also increases over time, albeit by a very small amount. This probably means there is some optimization we could have done here.

Interpolation of the large-scale data also takes quite a lot of time, compared to the shallow water simulation. Since the spatial interpolation only needs to be done every 30 seconds, this is has to be caused by the interpolation over time. This is one b-spline interpolation per cell per frame.

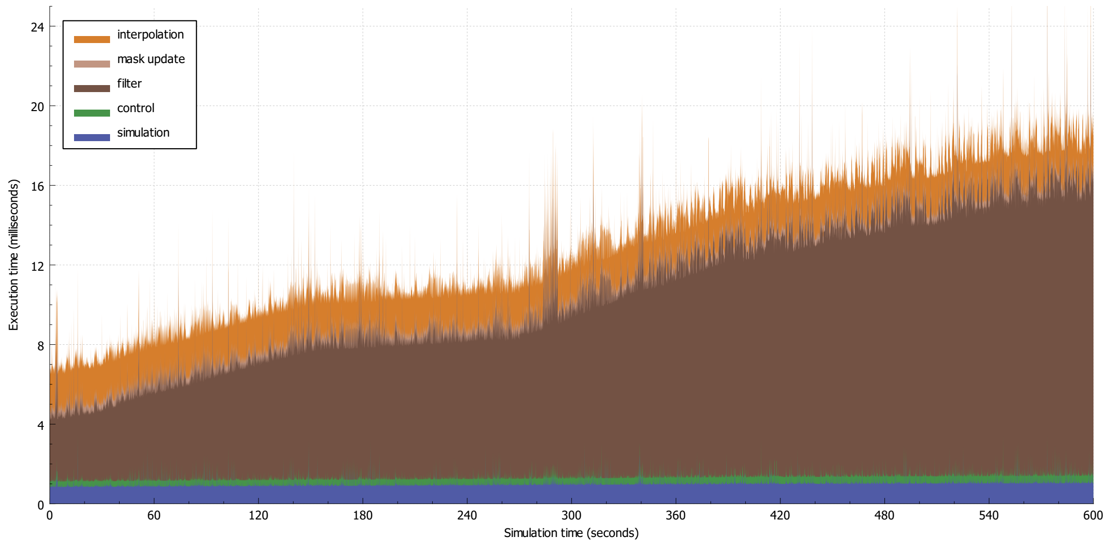
Execution time comparison

When comparing execution time for the different parameters, it is important to keep in mind that if the CPU is occupied with something else, even comparing results between different runs of the same simulation may not make sense. However, the execution time of the shallow water solver is practically constant. Since the different simulation results show a very similar execution time for the shallow water solver, we can compare them without any extra effort.

Varying α does not make a noticeable difference. Different cut-off frequencies, however, have a large impact on performance. See figures 5.13a and 5.13b, where the filter time for a lower cut-off frequency easily performs 4 times worse. The cause for this is clear: a lower cut-off frequency results in a wider kernel, which means convolution takes more time.



(a) Performance chart for $f_c = 0.125000$, $\alpha = 0.05$.



(b) Performance chart for $f_c = 0.015625$, $\alpha = 0.05$.

Figure 5.13.

5.6. Evaluation

This section evaluates on the results and the analysis. Criteria were established in section 5.1: realistic-looking water and controlled large-scale flow. Robustness of control and realistic looks were tested with the water drop scenario. Additionally, we compared the controlled simulation with an uncontrolled simulation from a similar origin.

In the comparison with the uncontrolled simulation (section 5.4.1), we saw the uncontrolled simulation is more turbulent and always flooding dry areas ahead of the controlled simulation. We also learned that the uncontrolled simulation was not exactly the same as the large-scale simulation (section 5.4.1). The controlled and uncontrolled simulations are quite similar, especially with weak control and a low cut-off frequency, where the standard deviation of the surface height error is between 0.09 and $0.36m$. This is really quite low, considering the size of a cell is already 6.25 meters wide and on average several times deeper. On the other hand, this is to be expected when the applied forces are similar. The relative error is as high as 13% and as low as 3%. This 3% is quite deceptive, however, as it is that low at the point in time where the range is highest, because of water streaming down into a dry lake. A similar thing goes for the 13%, which occurs when the uncontrolled simulation is stabilizing across the whole domain and thus the range is very low.

Overall, we believe the quality comparison results warrant using a low cut-off frequency with slightly stronger control for the surface height than for the velocities. This way, the flow of mass keeps up, but the velocities still have enough freedom to do what they should. The linear interpolation of the large-scale velocities is probably too simple.

Although the results favour weaker control, the differences between the different simulation comparisons are relatively small. A better argument comes from the water drop scenario. As the videos clearly showed, a lower cut-off frequency and weaker control give significantly longer-lasting waves. These videos also showed robustness of the controlled area. However, water drops landing on areas that should be dry caused these areas to remain wet for a very long time.

We mentioned robustness of the controlled area. Looking at the nature of linear interpolation, it is a given that large-scale errors are corrected quicker than the smaller waves. How much of this is due to the splitting of the frequencies is hard to tell, unfortunately.

While these results look good in the scenario the method was initially meant for, the scenarios are rather simple. Other scenarios would help evaluate the quality of the method, such as a scenario where water flows out of the domain, one where water is added on a slope or even a scenario where the small-scale bottom elevation works against the control data (ie. forcing to flow uphill).

Some of the problems created by these scenarios are tough to tackle - creating water on a dry cell is necessary in some situations, but unwanted in others. For example, we may need to spawn water on a slope where the cell may flow dry in a single frame (especially the case with weak control). At the same time, a scenario can occur where an incoming small-scale wave causes a controlled area to become dry for a short period. A typical scenario for this is a wave rolling on and off a beach.

Visually there is a lot of room for improvement: there is a lack of feedback for water depth and direction of flow. Visualizing water depth is mostly a matter of shading. Visualizing the flow can be done by advecting objects or textures with the water velocity, again a rendering thing.

The method performs in real-time, even with the widest kernel. This would still be the case if the entire domain was filled with water. However, the method scales upwards linearly, making larger simulations infeasible very quickly. A domain filled with cells of half the width would perform approximately 4 times slower. Control with the widest kernel would then execute in ≈ 80 milliseconds, which is only ≈ 12.5 frames per second, less than the 20 frames per second the simulation typically runs at. Optimization of the filter process is required before up-scaling is possible.

6. Conclusion

The objective of this thesis is to present a solution for visualizing detailed water flow under the constraints of a governing large-scale simulation in real-time.

First, a shallow water solver that can handle dry-wet border regions was introduced, combining techniques from two different publications. A small modification was made to the function that defines reflecting cell borders so that dry cells of equal height are also properly marked as reflective.

Then, the convolution process for separable filters was modified to ignore dry cells and reflecting borders. The choice for a Gaussian low-pass filter was motivated and the filter kernel was defined.

To obtain the high-frequency part of a signal, the low-pass filter result is subtracted from the original signal. The Gaussian filter now effectively acts as a high-pass filter. Repeated application of this high-pass filter caused numerical instabilities. The cause for these instabilities lay with truncation of the filter kernel.

Finally, each time step, the Gaussian high-pass filter is used to extract high-frequency information from the water state. This high-frequency information is added to interpolated data from the governing large-scale simulation. Together, they form the target state. The water state is then linearly interpolated with the target state. This allows the small-scale simulation to follow the governing large-scale simulation while allowing room for detailed water flow.

Results have shown that the quality is good, when looking at numerical error. The controlled simulation is very similar to an uncontrolled simulation that mirrors the governing large-scale simulation. Additionally, the simulation is robust to large interruptions, while still allowing smaller waves to disperse and dissolve naturally. Another advantage of the method is that interaction is easy to implement, as the underlying simulation is a simple shallow water simulation.

The largest downside of the technique is that it does not scale upwards easily. A wide filter kernel is required for the convolution process, which means quite a performance hit. Also, the method is really only suitable for situations that are suited for weak control and free boundaries. Strict control will mean a degradation of realism and have a bad impact on the duration high-frequency details are visible. Controlled boundaries will lead to unrealistic water fronts.

7. Future Work

A lot of work can be done to improve the current method. We will describe a problem we have not solved, as well as point out some areas of improvement.

- Unforeseen small-scale bottom elevation can block the flow of water (almost) completely, in which case our method will not follow the large-scale simulation correctly any more. Allowing water to be added into dry areas will solve this problem, but will create problems when controlled areas should become dry for other reasons. A related problem is that, currently, the initial water state must have water before anything can happen. A simple solution might be to check if there is water within a certain area.
- Investigation into a better advection scheme is useful, as the current semi-lagrangian advection scheme as introduced by [Sta99] introduces dissipative error, which tends to smear the fine details in the flow ([LO07]).
- Optimizing the convolution with the Gaussian will definitely improve performance. For example, the Gaussian can be approximated with box filters [BETVG08, S^+97] or calculated via integral images [Kov10].
- Investigation into a (different) better filter might benefit the quality or performance. Section 4.3 showed that we can deal with some error in the spatial domain, which means we do not necessarily need a perfect spatial response.
- Spatial interpolation of the large-scale data can maybe be improved. Perhaps a better method would be to start with nearest-neighbour interpolation and then perform convolution with a Gaussian filter, masking dry cells and reflecting edges.
- Visual feedback of the water depth and flow direction is poor. Combining with techniques like tiled directional flow [vH11] or animated textures generated with Fast Fourier Transforms [Tes04b] will solve these problems, while improving the rendering significantly with regard to visual quality. These animated textures could be precomputed or generated on the fly based on water depth and other properties. Alternatively, an animated texture could simply be scaled.
- A level-of-detail technique will be required to be able to scale up the simulation domain. A good start can be replacing the shallow water simulation in certain regions with a combination the techniques of [vH11] and [Tes04b]. This would be possible in regions where the water is deep enough for the bottom elevation not to influence the shallow water simulation strongly. The water height in these

regions will be almost flat. For an example of this combination of techniques, see appendix A.

- GPU enhancements will help improve performance.
- Better wet/dry boundary tracking is required to improve visual quality on areas that become dry.

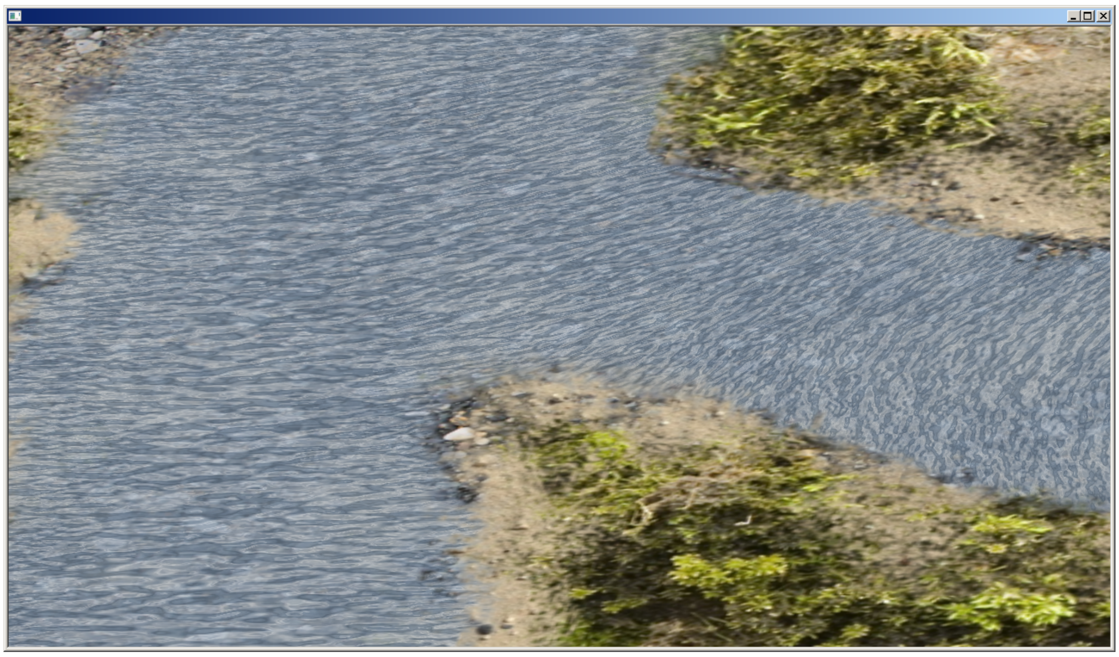


Figure A.1.: Tiled directional flow, as implemented by [vH11].

A. Side Project: Image-based Rendering

This appendix describes a simple combination of two techniques: tiled directional flow and FFT-based image generation. Appendices A.1 and A.2 will give a short overview of both techniques. Appendix A.3 will describe the combination of these techniques.

A.1. Tiled Directional Flow

Tiled directional flow by [vH11] is a technique used to visualize water flows (figure A.1). It can be completely implemented in a pixel shader. It works by splitting the domain into tiles (see figure A.2). For each tile, a normal map is sampled, which is scaled and moved according to data sampled from a texture that stores the flow. These normal maps are added together, after which the water surface can be shaded, resulting in waves that look animated.

The contribution factor of the overlapping tiles as shown in figure A.2 is decided by the center tile. It is strongest in the top-left corner and fades out towards the bottom-right corner. The author of [vH11] uses a cubic fall-off for this contribution factor, so that it

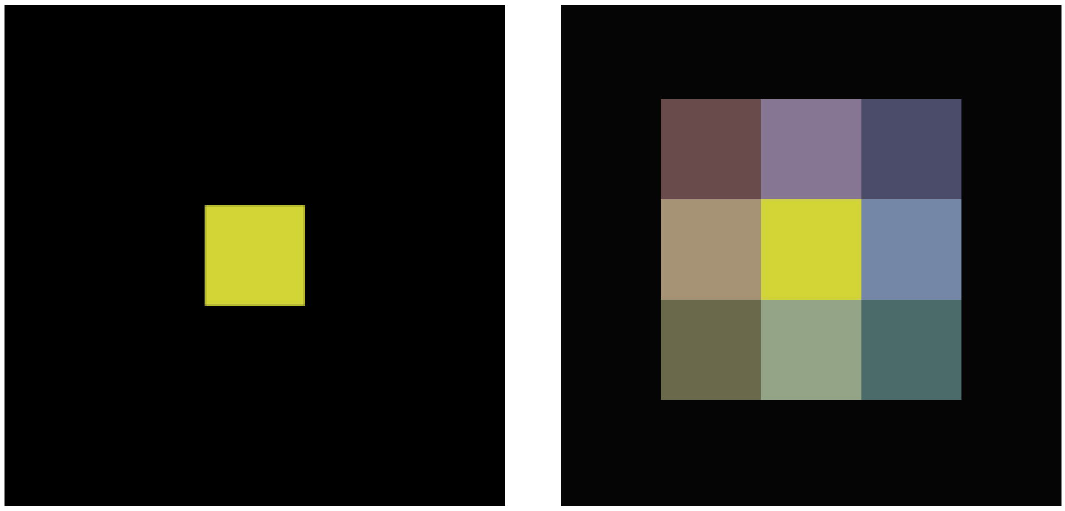

(a) Center tile. (b) Four large tiles overlapping the center tile.

Figure A.2.: Tiled directional flow tiling.

keeps a contribution close to 1 until close to the opposite edges, where it drops to 0.

The normal is computed as follows. First, the top two overlapping tiles' normal maps are added together. The normal of the left tile is multiplied with the contribution factor, the other's with 1 minus the contribution factor. The same for the bottom two tiles. The result of both additions is then added again in the same way, but the contribution is decided based on which tile addition is on top. The resulting normal is scaled by equation (A.1c). Finally, the normals are scaled by the transparency of the water as defined in the flow map, which is just another way of decreasing the normal strength where it would make sense (for example near the edges of the water).

$$Factor_x = \sqrt{(f(X)_x)^2 + (1 - f(X)_x)^2} \tag{A.1a}$$

$$Factor_y = \sqrt{(f(X)_y)^2 + (1 - f(X)_y)^2} \tag{A.1b}$$

$$NormalScaling = \frac{C}{Factor_x * Factor_y} \tag{A.1c}$$

Here, $f(X)$ is the contribution factor of the center tile X in x- and y-direction. $NormalScaling$ is the scaling factor for the final result. C is a user-defined constant, again to tweak the strength of the waves.

A.2. Fast Fourier Transform

FFT-based water animation [Tes04b, JG01] is core to many water rendering techniques, even if just for adding details or ambient waves. It generates animated waves given a phase/amplitude spectrum. It allows for the user to define both a spectrum, which encodes wave amplitudes and phases, and a dispersion relation, which relates water depth and wavelength to wave speed. The spectrum can be modified in such a way that it emphasizes waves in a certain direction more than others. Additionally, it can be modified to show choppy waves. The result is a realistic water animation that can be stored as a heightmap or normal map.

The basis of this technique is the fast Fourier transform, which is an efficient way of evaluating the following sum:

$$h(\vec{x},t) = \sum_k \tilde{h}(\vec{k},t)\exp(i\vec{k}\cdot\vec{x}) \tag{A.2}$$

Here, $h(\vec{x},t)$ is the height at position $\vec{x}$ at time t. $\vec{k}$ is the wave vector, which is a composition of the wave numbers in x- and y-direction. A wave number k relates to the wavelength λ as follows: $k = 2\pi/\lambda$. $\tilde{h}(\vec{k},t)$ is the representation of the sine wave of wave number k as a complex number. It encodes the phase and amplitude. $\exp(i\vec{k}\cdot\vec{x}$ represents a sine wave with wavelength $2\pi/k$, phase 0 and amplitude 1, sampled at position $\vec{x}$. Multiplied with $\tilde{h}(\vec{k},t)$, it results in a sine wave with wavelength $2\pi/k$ and a phase and amplitude as defined by $\tilde{h}$ at time t. Adding all the sine waves together gives the height of the water at the requested position. Sampling this function for an entire grid, given the right spectrum, results in an image of waves.

More details regarding the FFT method can be found in [Tes04b] and [JG01].

A.3. Tiled FFT

As an exploration in image-based rendering, we combined the previous two techniques into one. Simply put, we generate an animated normal map and use tiled directional flow to sample it instead of the static normal map. See figure A.3 for a screenshot.

In contrast to the default tiled directional flow, we do not move the normal map over time. Instead, water movement is generated by modifying the spectrum to generate waves in the direction we desire. We used the Phillips spectrum (equation (A.3), [Tes04b]), which generates waves based on wind velocity. For details on how to integrate this with the FFT-based algorithm, see [Tes04b, JG01].

$$P_h(\vec{k}) = A\frac{\exp\left(-1/(kL)^2\right)}{k^4}|\vec{k}\cdot\vec{w}|^2 \tag{A.3}$$

Here, $\vec{k}$ is the wave vector, k is the wave vector magnitude and $\vec{w}$ is the wind velocity. We modify this spectrum slightly by multiplying negative outcomes of the dot product with a scaling factor, to reduce the strength of waves going in the opposite direction. We found that they can be eliminated completely, giving a subjectively better result.

To generate an animation map with moving waves, we simply set the wind speed. This wind speed does not need to be high to create water movement required for calm flowing water (our wind speed had a magnitude of 1). To get waves in the direction of the flow, we sample the normal map with the right orientation. The result can be viewed at `https://www.youtube.com/watch?v=rF_cAAwXlz4`.

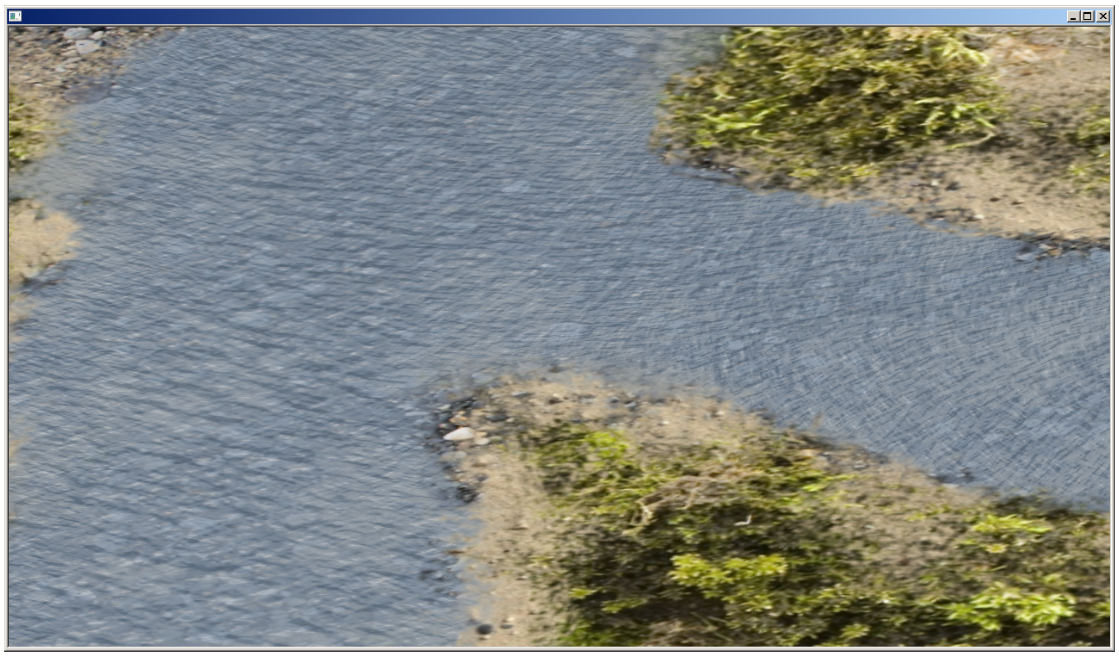


Figure A.3.: Tiled directional flow with animated normal maps, generated with the FFT.

The combination of these two techniques is interesting, because it is highly customizable and highly realistic at the same time, given the right spectrum. The required animated normal maps are small enough to be generated in real-time. Alternatively, a large number of animated maps could be precomputed. Interesting parameters to vary would for example be wind speed and water depth. Water depth is a parameter of the dispersion relation, for which several versions exist. We used the following dispersion relation [Tes04b]:

$$\omega(\vec{k}) = \sqrt{gk\tanh(kd)} \tag{A.4}$$

Here,k is the wave vector magnitude, g is gravity and d is the water depth.

Finding a realistic spectrum, for example one that uses flow velocity instead of wind velocity, might require some research, as the Phillips spectrum describes open ocean waves generated by wind. Scenarios like a flood or a flowing river have significant influences resulting from bottom elevation and flow speed, which could result in a spectrum significantly different from a wind-based spectrum.

Bibliography

[AGJN02] Arul Prakash Asirvatham, AP Gachibowli, CV Jawahar, and PJ Narayanan. Script segmentation of multi-script documents. 2002. 17

[APKG07] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J Guibas. Adaptively sampled particle fluids. In *ACM Transactions on Graphics (TOG)*, volume 26, page 48. ACM, 2007. 7

[BETVG08] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008. 17, 59

[BHN07] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. In *ACM Transactions on Graphics (TOG)*, volume 26, page 46. ACM, 2007. 8

[bri] Fluid simulation: Siggraph 2007 course notes. 12

[BSL07] R Byron Bird, Warren E Stewart, and Edwin N Lightfoot. *Transport phenomena*. John Wiley & Sons, 2007. 13

[CBP] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 219–228. 7

[CFL$^+$07] Nuttapong Chentanez, Bryan E Feldman, François Labelle, James F O'Brien, and Jonathan R Shewchuk. Liquid simulation on lattice-based tetrahedral meshes. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 219–228. Eurographics Association, 2007. 8

[CM10] Nuttapong Chentanez and Matthias Müller. Real-time simulation of large bodies of water with small scale details. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 197–206. Eurographics Association, 2010. 8, 12, 14, 15, 16

[CM11] Nuttapong Chentanez and Matthias Müller. Real-time eulerian water simulation using a restricted tall cell grid. In *ACM Transactions on Graphics (TOG)*, volume 30, page 82. ACM, 2011. 8

[Cor08] Hilko Cords. Moving with the flow: Wave particles in flowing liquids. 2008. 7

[CS09] Hilko Cords and Oliver G Staadt. Real-time open water environments with interacting objects. In *NPH*, pages 35–42. Citeseer, 2009. 8

[CZY11] Fan Chen, Ye Zhao, and Zhi Yuan. Langevin particle: A self-adaptive lagrangian primitive for flow simulation enhancement. In *Computer Graphics Forum*, volume 30, pages 435–444. Wiley Online Library, 2011. 7

[Day09] Mike Day. Insomniac's water rendering system, 2009. `http://www.insomniacgames.com/tech/articles/0409/files/water.pdf`. 8

[EQYF13] R Elliot English, Linhai Qiu, Yue Yu, and Ronald Fedkiw. Chimera grids for water simulation. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 85–94. ACM, 2013. 8

[HNC02] Damien Hinsinger, Fabrice Neyret, and Marie-Paule Cani. Interactive animation of ocean waves. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 161–166. ACM, 2002. 8

[JG01] Lasse Staff Jensen and Robert Golias. Deep-water animation and rendering. In *Game Developers Conference (Gamasutra)*, 2001. 62, 63

[JKS95] Ramesh Jain, Rangachar Kasturi, and Brian G Schunck. *Machine vision*, volume 5. McGraw-Hill New York, 1995. 17

[Kal08] Daniel Kallin. Real-time large scale fluids for games. *SIGRAD 2008*, page 31, 2008. 8

[KO96] S Koshizuka and Y Oka. Moving-particle semi-implicit method for fragmentation of incompressible fluid. *Nuclear science and engineering*, 123(3):421–434, 1996. 7

[Kov10] Peter Kovesi. Fast almost-gaussian filtering. In *Digital Image Computing: Techniques and Applications (DICTA), 2010 International Conference on*, pages 121–125. IEEE, 2010. 17, 59

[KP$^+$07] Alexander Kurganov, Guergana Petrova, et al. A second-order well-balanced positivity preserving central-upwind scheme for the saint-venant system. *Communications in Mathematical Sciences*, 5(1):133–160, 2007. 8

[KW06] Peter Kipfer and Rüdiger Westermann. Realistic and interactive simulation of rivers. In *Proceedings of Graphics Interface 2006*, pages 41–48. Canadian Information Processing Society, 2006. 7

[LO07] Richard Lee and Carol O'Sullivan. A fast and compact solver for the shallow water equations. In *VRIPHYS*, pages 51–57, 2007. 8, 59

[LTKF08] Frank Losasso, Jerry O Talton, Nipun Kwatra, and Ron Fedkiw. Two-way coupled sph and particle level set fluid simulation. *Visualization and Computer Graphics, IEEE Transactions on*, 14(4):797–804, 2008. 7

[Luc14] Steven J Luck. *An introduction to the event-related potential technique.* MIT press, 2014. 26, 31

[MCG03] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159. Eurographics Association, 2003. 7

[MM13] Miles Macklin and Matthias Müller. Position based fluids. *ACM Transactions on Graphics (TOG)*, 32(4):104, 2013. 7

[Mon94] Joe J Monaghan. Simulating free surface flows with sph. *Journal of computational physics*, 110(2):399–406, 1994. 7

[MSJT08] Matthias Müller, Jos Stam, Doug James, and Nils Thürey. Real time physics: class notes. In *ACM SIGGRAPH 2008 classes*, page 88. ACM, 2008. 12, 13, 14, 15

[MTPS04] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. In *ACM Transactions On Graphics (TOG)*, volume 23, pages 449–456. ACM, 2004. 9

[NB11] Michael B Nielsen and Robert Bridson. Guide shapes for high resolution naturalistic liquid simulation. In *ACM Transactions on Graphics (TOG)*, volume 30, page 83. ACM, 2011. 9

[NSCL08] Rahul Narain, Jason Sewall, Mark Carlson, and Ming C Lin. Fast animation of turbulence using energy transport and procedural synthesis. In *ACM Transactions on Graphics (TOG)*, volume 27, page 166. ACM, 2008. 8

[Ott11] Björn Ottosson. *Real-time Interactive Water Waves.* PhD thesis, Masters thesis, School of Engineering Physics, Royal Institute of Technology, 2011. URL http://www. nada. kth. se/utbildning/grukth/exjobb/rapportlistor/2011/rapporter11/ottosson_bjorn_11105. pdf, 2011. 8

[PHT$^+$13] Zherong Pan, Jin Huang, Yiying Tong, Changxi Zheng, and Hujun Bao. Interactive localized liquid motion editing. *ACM Transactions on Graphics (TOG)*, 32(6):184, 2013. 9

[PTSG09] Tobias Pfaff, Nils Thuerey, Andrew Selle, and Markus Gross. Synthetic turbulence using artificial boundary layers. *ACM Transactions on Graphics (TOG)*, 28(5):121, 2009. 8

[RTWT12] Karthik Raveendran, Nils Thuerey, Chris Wojtan, and Greg Turk. Controlling liquids using meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 255–264. Eurographics Association, 2012. 9

[S$^+$97] Steven W Smith et al. The scientist and engineer's guide to digital signal processing. 1997. 17, 18, 26, 32, 33, 59

[Sta99] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999. 8, 14, 59

[Tes04a] Jerry Tessendorf. Interactive water surfaces. *Game Programming Gems*, 4:265–274, 2004. 8

[Tes04b] Jerry Tessendorf. Simulating ocean water. 2004. 8, 59, 62, 63, 64

[TKPR09] Nils Thürey, Richard Keiser, Mark Pauly, and Ulrich Rüde. Detail-preserving fluid control. *Graphical Models*, 71(6):221–228, 2009. 9

[TLP06] Adrien Treuille, Andrew Lewis, and Zoran Popović. Model reduction for real-time fluids. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 826–834. ACM, 2006. 8

[Tur07] K Turkowski. Incremental computation of the gaussian. *GPU Gems*, 3, 2007. 19

[vH11] Frans van Hoesel. Tiled directional flow. In *ACM SIGGRAPH 2011 Posters*, page 19. ACM, 2011. 9, 59, 61

[Vla10] Alex Vlachos. Water flow in portal 2. *SIGGRAPH Course on Advances in Real-Time Rendering in 3D Graphics and Games*, 2010. 9

[VW02] Jarke J Van Wijk. Image based flow visualization. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 745–754. ACM, 2002. 9

[WST09] Martin Wicke, Matt Stanton, and Adrien Treuille. Modular bases for fluid dynamics. In *ACM Transactions on Graphics (TOG)*, volume 28, page 39. ACM, 2009. 8

[YHK07] Cem Yuksel, Donald H House, and John Keyser. Wave particles. In *ACM Transactions on Graphics (TOG)*, volume 26, page 99. ACM, 2007. 7

[YNBH09] Qizhi Yu, Fabrice Neyret, Eric Bruneton, and Nicolas Holzschuch. Scalable real-time animation of rivers. In *Computer Graphics Forum*, volume 28, pages 239–248. Wiley Online Library, 2009. 9

[YNBH11] Qizhi Yu, Fabrice Neyret, Eric Bruneton, and Nicolas Holzschuch. Lagrangian texture advection: Preserving both spectrum and velocity field. *Visualization and Computer Graphics, IEEE Transactions on*, 17(11):1612–1623, 2011. 9