
Master Informatique

---

**Projet IMA**
**Mid-Term Report**
**Real Time Fluid Simulations**

---

**Étudiant:** Basci Onur
**N°:** 21309649

# Contents

**1 Context** **2**

**2 Introduction** **3**
2.1 Navier-Stokes Equations . . . . . 3
2.2 States of Matters . . . . . 3

**3 State of the Art** **4**
3.1 Procedural Water . . . . . 4
3.2 Height Field Approximation . . . . . 5
3.3 Marker and Cell Method . . . . . 6
3.4 Adaptive Grids . . . . . 8
3.5 Smoothed Particle Hydrodynamics . . . . . 9
3.6 Position Based Fluids . . . . . 9

**4 Development Tools** **10**
4.1 Unity . . . . . 10

**5 What has been done until now?** **10**

**6 What's next?** **12**
6.1 schedule . . . . . 12

**7 Bibliography** **12**

# 1 Context

Over the past four decades, fluid simulation has captivated researchers due to its intricate and dynamic nature. These simulations find frequent application in the realm of filmmaking and animation, as showcased in works such as "The Day After Tomorrow" and "The Good Dinosaur." Leveraging high-performance computer systems and offline simulation techniques, these productions can generate highly detailed simulations without time constraints, unlike real-time simulations. Real-time simulations face two primary challenges: time and resources. To meet industry standards for video games, real-time simulations must be calculated swiftly to achieve playback rates of at least 30 or 60 frames per second. This necessitates all computations for each time period of the simulation to be completed within 0.03 or 0.016 seconds. Additionally, real-time simulations for video games must contend with the limitations of consumer-grade hardware, as they are intended for use on personal computers.

Advancements in computer graphics technology, particularly in GPU (Graphics Processing Unit) and parallel computing, are narrowing the gap between real-time and offline simulations. Researchers are delving into innovative methods to optimize these simulations, balancing the constraints imposed while retaining the complex structure of fluids.

Throughout this project, we aim to delve into various methods put forth by researchers and implement one of the latest and most efficient techniques available. Our focus will be on utilizing Unity, a game engine renowned for offering numerous optimization tools, including compute shaders for GPU utilization, the Job System, and the Burst compiler. Additionally, the simulation should be highly parameterizable since we are working on fluids and not only liquids. In fact, a fluid can be water and gas; therefore, we aim to have a simulation model that is capable of simulating phase transitions.

# 2 Introduction

## 2.1 Navier-Stokes Equations

A large number of water simulations rely on the Navier-Stokes Equations, which are a set of partial differential equations describing the motion of viscous fluid substances. They are typically written as:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u} \tag{1}$$

$$\nabla \cdot \vec{u} = 0 \tag{2}$$

Where $\vec{u}$ represents the velocity field, $p$ represents the pressure field, $\vec{g}$ denotes external forces (such as acceleration due to gravity), $\nu$ represents the kinematic viscosity, and $\rho$ represents the density.

The first equation is essentially derived from Newton's second law, $F = ma$. Here, $\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u}$ represents the material derivative, corresponding to acceleration. The right-hand side of the equation corresponds to $F$, where $\nabla p$ represents the force resulting from pressure; high-pressure regions push on lower-pressure regions. The term $\nu \nabla^2 \vec{u}$ corresponds to the force resulting from viscosity, interpreted as a force attempting to align the particle's velocity with the average velocity of nearby particles. Finally, we can define mass as $m = \frac{\rho}{V}$, thus we get the Navier-Stokes equations with some simplifications.

The second equation pertains to incompressibility. During these simulations, we assume water is incompressible, which is reasonable since water in real life is very close to being incompressible. Unfortunately, we cannot make the same assumption for gas, as it has a much lower density. For water, the density is roughly $1000\,\mathrm{kg/m^3}$, and for air, it's roughly $1.3\,\mathrm{kg/m^3}$, a ratio of about $700 : 1$. [1]

The main idea behind most physics-based water simulations is to solve these equations for the velocity and pressure fields or simulate the effects resulting from these equations. But before we discuss methods of simulation, let's delve into the states of matter, which are crucial for simulating different states in our simulation.

## 2.2 States of Matters

The primary states of matter are commonly classified into four categories: solid, liquid, gas, and plasma. Each state exhibits unique properties and behaviors, which are governed by the interactions between particles and the environment.

- In a solid, constituent particles (ions, atoms, or molecules) are closely packed together. The forces between particles are so strong that the particles cannot move freely but can only vibrate. As a result, a solid has a stable, definite shape, and a definite volume. Solids can only change their shape by an outside force, such as when broken or cut.
- A liquid is a nearly incompressible fluid that conforms to the shape of its container but retains a (nearly) constant volume. The volume is definite if the temperature and pressure are constant. When a solid is heated above its melting point, it becomes liquid, given that the pressure is higher than the triple point of the substance.

- A gas is a compressible fluid. Not only will a gas conform to the shape of its container, but it will also expand to fill the container. In a gas, the molecules have enough kinetic energy so that the effect of intermolecular forces is small (or zero for an ideal gas), and the typical distance between neighboring molecules is much greater than the molecular size. A gas has no definite shape or volume but occupies the entire container in which it is confined. A liquid may be converted to a gas by heating at constant pressure to the boiling point, or else by reducing the pressure at constant temperature. At temperatures below its critical temperature, a gas is also called a vapor and can be liquefied by compression alone without cooling.

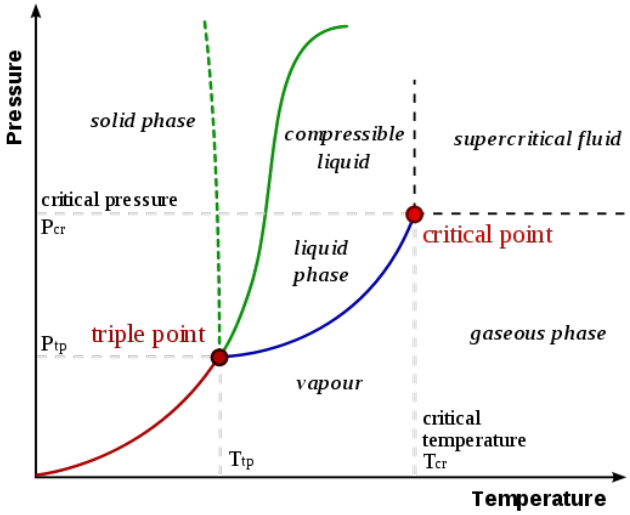


Figure 1: Phase Diagram

So, as evident from the phase diagram, phase transitions primarily depend on two variables: temperature and pressure. Consequently, our simulation must be parametrizable with these two variables.

# 3 State of the Art

## 3.1 Procedural Water

A procedural approach directly animates a physical effect instead of simulating its underlying cause. For instance, a common technique to generate the surface of bodies of water like lakes or oceans procedurally involves overlaying sine waves of varying amplitudes and directions. [3] Procedural animation techniques offer limitless creativity, allowing for the creation of desired visual effects without constraints. Controllability is a key advantage of procedural animation, particularly valuable in gaming. However, a drawback of procedurally simulating water is the challenge of accurately depicting its interaction with boundaries or immersed objects.

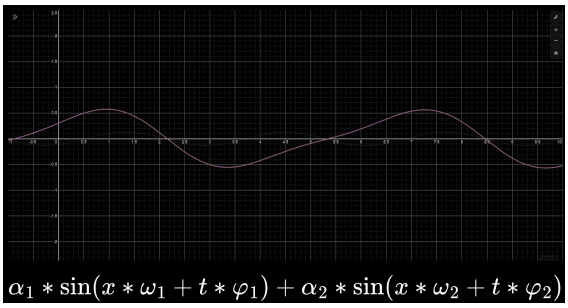


Figure 2: An example of sum of sinus with different amplitude and frequencies

Figure 3: A result obtained from the article A simple Model of ocean waves

Although the sum of sinus method is highly efficient and can simulate large amounts of water, such as ocean waves, it has some drawbacks. Firstly, it only represents the surface of the water and does not account for the volume beneath. Secondly, it does not interact with its environment, such as objects or boundaries within the water. Finally, it is only applicable for liquid simulation and cannot accurately represent other states of matter or their interactions.

## 3.2 Height Field Approximation

Another method that attempts to simulate the surface of water is the Height Field Approximation. If our interest lies solely in simulating the water surface, simulating the entire three-dimensional body of water would be unnecessary. The concept of the Height Field approach involves representing the water as a set of height blocks. This can be visualized as a 2D array where the index represents positions and the corresponding value represents the height of the block.

To simulate the movement of these blocks, we utilize Archimedes' principle. According to this principle, the upward buoyant force exerted on a body immersed in a fluid, whether fully or partially, is equal to the weight of the fluid displaced by the body. By applying this principle, we can calculate the acceleration of each block based on the difference in height between the block and its neighbors. [1]

This method is pretty simple to simulate and it is quite fast. We can also simulate the interaction with the objects still with the Archimedes's principal. But due to its structure it can not provide overturning waves and splashes.

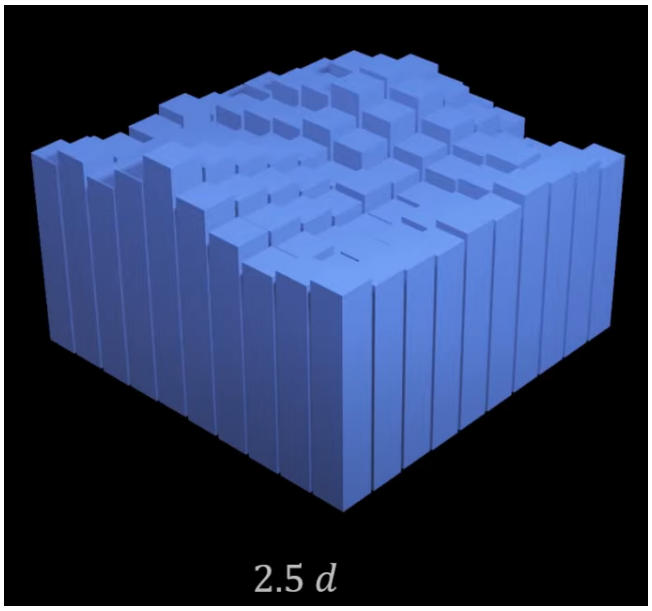


Figure 4: A representation of the height field approach

### 3.3 Marker and Cell Method

The marker and cell method involves discretizing 2D or 3D space to solve the Navier-Stokes equation. The space is represented as a staggered grid where each block contains information such as pressure, velocity, viscosity, etc. Velocity values are stored on the edges rather than the center of the block. This data structure is typically preferred for vector field representation as it allows for more accurate derivative calculations. The concept is to solve partial differential equations to calculate the velocity and pressure values for each block.

- Start with an initial divergence-free velocity field $\vec{u}(0)$.
- For time step $n = 0, 1, 2, \ldots$
- Determine a good time step $\Delta t$ to go from time $t_n$ to time $t_{n+1}$.
- Set $\vec{u}_A = \text{advect}(\vec{u}_n, \Delta t, \vec{u}_n)$.
- Add $\vec{u}_B = \vec{u}_A + \Delta t \vec{g}$.
- Set $\vec{u}_{n+1} = \text{project}(\Delta t, \vec{u}_B)$.

[1]

Here, the "advect" method is utilized to calculate velocity advection. This process involves Semi-Lagrangian advection. In the second step of the for loop, external forces are applied. This step employs a simple Forward Euler update. Lastly, the "project" method is focused on ensuring the water remains incompressible. It achieves this by subtracting the pressure gradient from the intermediate velocity field $\vec{u}$.

$$\vec{u}_{n+1} = \vec{u} - \frac{\Delta t}{\rho} \nabla p$$

With the constraint that

$$\nabla \cdot \vec{u}_{n+1} = 0$$

By using the central difference approximation then putting the velocity update equation on the divergence free equation we obtain the Poisson equation defined as

$$-\frac{\Delta t}{\rho}\nabla \cdot \nabla p = -\nabla \cdot \vec{u}$$

[1]

So the purpose of the project part is to solve the Poisson equation and then use the pressure field to update the velocity to make it divergence free.

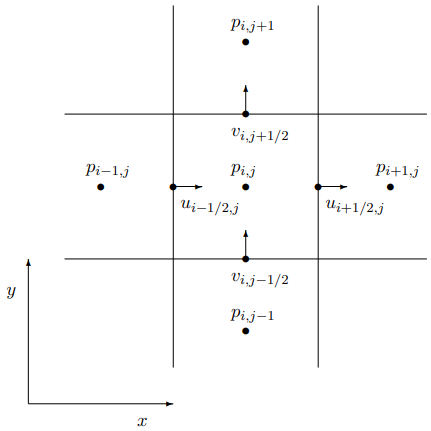


Figure 2.1: The two-dimensional MAC grid.

Figure 5: Staggered grid in 2D

The Marker and Cell method indeed produces realistic results and can interact with the environment by designating certain cells as obstacle solid cells. However, due to the high computational cost associated with calculating pressure and vector fields for each block, this method is not feasible for real-time simulations. This leads us to the question: can we employ a different data structure that requires less computation while still capturing the complex structure of the water? This will be explored in our next section.

## 3.4 Adaptive Grids

An approach to address the complexity of resolving the Navier-Stokes equation on the grid is to employ an adaptive grid. An adaptive grid dynamically adjusts the resolution of the simulation grid based on the complexity and movement of the fluid flow. This means that regions with high fluid complexity or where fine details are important will have a higher grid resolution, while regions with relatively simple fluid behavior can use a coarser grid. By adaptively adjusting the grid resolution, computational resources can be focused where they are most needed, thereby improving the efficiency of the simulation.

The article "Real-Time Eulerian Water Simulation Using a Restricted Tall Cell Grid" [2] proposes a state-of-the-art method for adaptive grid-based real-time water simulation. The main contributions of the article are as follows:

- A tall cell grid data structure that allows for efficient liquid simulation within a wide variety of scenarios.
- An efficient multigrid Poisson solver for the tall cell grid. The solver can also be used to accelerate fluid simulations on the
- Several modifications in the level set and velocity advection schemes that allow both larger time steps to be used and an efficient GPU implementation.

The main idea of the adaptive grid is to integrate a generalized height field representation with a three-dimensional grid layered on top of it. This allows us to approximate the general behavior of the fluid using tall cells while still capturing the interesting features of a full 3D simulation, such as splashes and overturning. Additionally, the article imposes certain constraints to simplify the data structure and algorithm, as well as making the method GPU-friendly.

- Each water column contains exactly one tall cell
- The tall cell is located at the bottom of the water column
- Velocities are stored at the cell center for regular cells and at the top and bottom of tall cells

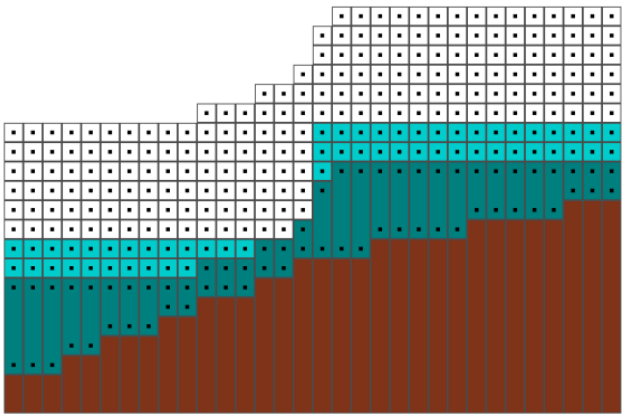

Figure 6: A represantation of the tall cell grid

Even though the article claims to achieve efficient results in various scenarios, maintaining at least 30 frames per second, the method it presents is highly restricted. The version proposed

in the article does not appear to be configurable for parameters such as pressure and viscosity of the liquid. Therefore, we will explore more parameterizable methods, such as Smoothed Particle Hydrodynamics.

## 3.5 Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) represents fluids as a collection of particles distributed throughout the domain, with each particle carrying properties such as position, velocity, density, and other physical quantities. SPH employs kernel functions to evaluate spatially varying quantities at any point within the fluid domain, defining the influence of nearby particles on a given particle based on their distances. Through interpolation, the properties of a fluid at any point are computed by interpolating the values of neighboring particles using these kernel functions. A popular choice for a kernel function is the poly6 kernel. [1]

$$W_{poly6}(r) = \frac{315}{64\pi d^9} \begin{cases} (d^2 - r^2)^3 & \text{if } 0 \leq r \leq d \\ 0 & \text{otherwise} \end{cases}$$

For example, we can calculate the smooth density field from the individual positions and masses of the particles with

$$\rho(x) = \sum_j m_j W(|x - x_j|).$$

From this density field we can also compute an arbitrary smoothed field $A_s$ with

$$A_s(x) = \sum_j m_j \frac{A_j}{\rho_j} W(|x - x_j|) \tag{3}$$

A nice property of this formulation is that the gradient of such a field can easily be computed by replacing the kernel by the gradient of the kernel

$$\nabla A_s(x) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(|x - x_j|) \tag{4}$$

Indeed, the main idea behind Smoothed Particle Hydrodynamics (SPH) is to apply the SPH rule to the pressure or viscosity term and update particle properties accordingly. One of the main advantages of this method is that since we work with particles, mass conservation is guaranteed, and as we move the particles directly, all properties are advected automatically.

Additionally, another advantage of SPH is that from the governing equation(3) we can calculate any field that we desire. This makes SPH highly configurable and a promising choice for simulating liquids.

However, SPH is sensitive to density fluctuations resulting from neighborhood deficiencies, and enforcing incompressibility can be computationally expensive due to the unstructured nature of the model. Although recent advancements have improved efficiency by an order of magnitude, small time steps remain a requirement, limiting its applicability in real-time simulations.[4]

## 3.6 Position Based Fluids

In SPH algorithms, instability can arise if particles lack sufficient neighbors for accurate density estimates. Typical solutions involve taking sufficiently small time steps or ensuring a sufficient number of particles. However, these solutions are often impractical for real-time simulations.

The article "Position Based Fluids" [4] proposes a different approach to maintain stability while ensuring a stable density. This approach introduces a position constraint for each particle. The constraint is defined as:

$$C_i(p_1, \ldots, p_n) = \frac{\rho_i}{\rho_0} - 1 \tag{5}$$

where $\rho_0$ is the rest density and $\rho_i$ is given by the standard SPH density estimator:

$$\rho_i = \sum_j m_j W(\mathbf{p}_i - \mathbf{p}_j, h) \tag{6}$$

The idea is to find the $\Delta p$ that satisfies $C(p + \Delta p) = 0$ and than modify the position of the particle with $\Delta p$. The method seems to work very efficiently. The article says that it can simulate 128K of particles with the time step of 4.2 ms. Furthermore it still lets us modify the pressure since $\rho_0$ is configurable.

It seems like this method satisfy all of our requirements and seems applicable during this project.

# 4 Development Tools

## 4.1 Unity

During this project I will be working on Unity which is a very popular game engine with many optimization features. One of my objectif is to learn these features. Let's talk about some of them.

- **Compute** shaders are shader programs that run on the GPU, outside of the normal rendering pipeline. They can be used for massively parallel General-purpose computing on graphics processing units algorithms, or to accelerate parts of game rendering.
- **Unity's Job System** is a feature introduced to Unity game engine to enable efficient multi-threaded processing of tasks. It allows developers to take advantage of multi-core processors without the complexity of manual thread management. The Job System works in conjunction with Unity's Entity Component System (ECS) and Burst Compiler to achieve high-performance computing.
- **Unity's Burst Compiler** is a high-performance compiler specifically designed to optimize C code for running on multiple platforms, particularly targeting CPUs. It's primarily used in Unity game development to achieve significant performance improvements, especially for CPU-bound tasks. Burst Compiler works in conjunction with Unity's Job System and Entity Component System (ECS) to maximize performance gains.

# 5 What has been done until now?

In the first part of the project, I began with some revision on vector calculus and fluid dynamics since it is necessary to understand most of the articles. Then, I conducted a state-of-the-art research where I analyzed different methods to find the most suitable solution to our problem. Next, I worked on a Marker and Cell simulation to better understand the functioning of a basic fluid simulation.

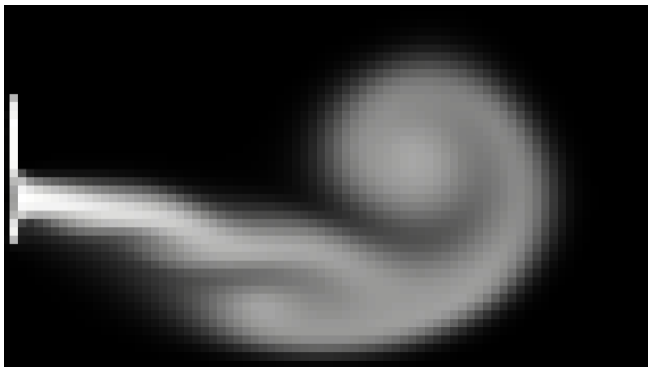

Figure 7: Marker and cell method to simulate Smoke

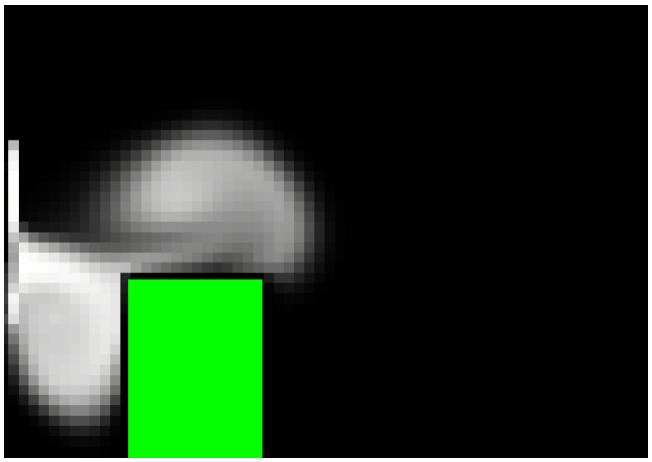

Figure 8: Smoke simulation with an obstacle interaction

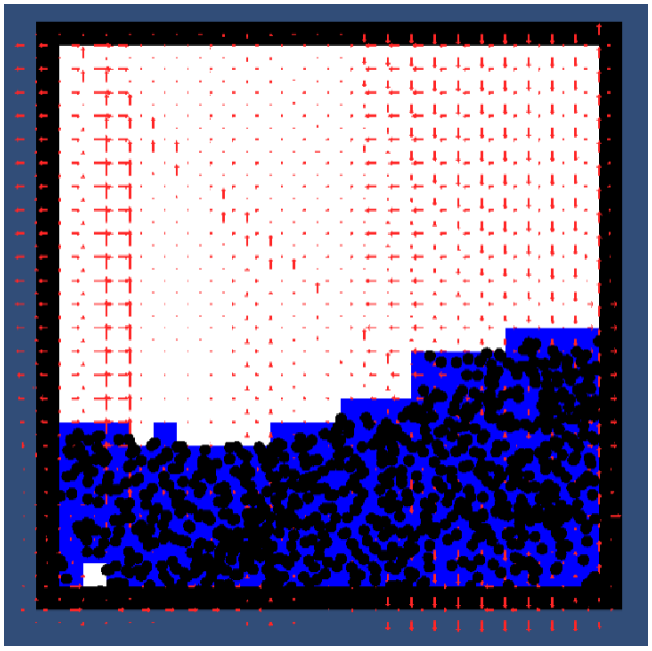

Figure 9: Marker and cell method with Marker Particles

Firstly, I adapted the simulation to render smoke by simply advecting the blocks containing it with Semi-Lagrangian advection. Then, I modified it to render water by using Marker Particles. Here are some pictures from my simulations:

For the first simulation, I used a grid resolution of $100 \times 50$, and for the second one, I used a grid resolution of $25 \times 25$. Both simulations achieved more than 30 frames per second (fps). However, when I doubled the resolution, I observed approximately 15 fps for the smoke simulation and only 0.2 fps for the water simulation.

# 6 What's next?

## 6.1 schedule

Here is the provisional schedule of things to do up to the end of the project

- Week 18-24 March: Implementation of the Smoothed Particle Hydrodynamics
- Week 25-31 March: Implementation of the Position Based Fluids
- Week 1-7 April: Work on the Optimization of the methods
- Week 22-28 April: Work on the rendering technics for Fluid and Smoke
- Week 29 April-5 May: Experimentation and Testing

# 7 Bibliography

# References


[1] Robert Bridson and Matthias Müller-Fischer. Fluid simulation: Siggraph 2007 course notes video files associated with this course are available from the citation page. In *ACM SIGGRAPH 2007 courses*, pages 1–81. 2007.

[2] Nuttapong Chentanez and Matthias Müller. Real-time eulerian water simulation using a restricted tall cell grid. In *ACM Siggraph 2011 Papers*, pages 1–10. 2011.

[3] Alain Fournier and William T Reeves. A simple model of ocean waves. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 75–84, 1986.

[4] Miles Macklin and Matthias Müller. Position based fluids. *ACM Transactions on Graphics (TOG)*, 32(4):1–12, 2013.