UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

LIMBIC Entertainment

Real-time rendering and plausible simulation of oceans

Masterarbeit

zur Erlangung des Grades Master of Science (M.Sc.)
im Studiengang Computervisualistik

vorgelegt von

Nico Ell

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Dr.-Ing. Florian Mehm (Limbic Entertainment GmbH)

Koblenz, im September 2020

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☒	☐

Bad Homburg, 30.09.2020 .... [illegible]

(Ort, Datum) (Unterschrift)

Acknowledgement

First, I would like to thank my supervisor, Prof. Dr.-Ing. Stefan Müller, for his continuing great efforts, to shape the course of study computational visualistics. Computer graphics became my passion, and I could not have studied anything better.

Then, my thanks go to *Limbic Entertainment* and my advisor, Dr.-Ing. Florian Mehm, for all the support and the opportunity to write this thesis in collaboration even during the difficult situation of a global pandemic.

A big thank-you goes to my wife, Nicha, for her invaluable support, her patience and her ongoing endeavour of teaching me comma placement.

Last but not least, I thank my friends Richard Gottschalk, Dominik Stowasser and Christian Schmitt for their suggestions and expertise.

Kurzfassung

Ein Ozean, der unterschiedliche Windbedingungen widerspiegelt, sich entsprechend dem Terrain verhält und Interaktionen mit Objekten ermöglicht, hat großes Potenzial, die Glaubwürdigkeit von virtuellen Welten zu steigern. Dessen Simulation und Darstellung muss jedoch schnell genug sein, um auch gemeinsam mit weiteren Elementen der Welt in Echtzeit berechnet werden zu können. Diese Arbeit verwendet ein GPU-gestütztes Verfahren zum Generieren der Oberfläche tiefer Gewässer mithilfe der schnellen Fourier-Transformation. Es wird beschrieben, wie die dabei verwendeten Wellenspektren abgetastet werden können, um nahtlose Übergänge bei wechselndem Seegang zu gewährleisten. Fehlende Einflüsse des Terrains und der Interaktion mit Objekten werden durch eine Flachwassersimulation mit der Lattice-Boltzmann-Methode ergänzt. Dabei wird eine jüngst vorgestellte Variante mit geringerer Speichernutzung erprobt und gezeigt, dass ihre Eigenschaften es erlauben, ein sehr großes Gebiet mit variablem Detailgrad abzudecken. Die Erkenntnisse können auch außerhalb der Computergrafik zum Einsatz kommen. Eine abschließende Performance-Analyse und Diskussion bestätigt die Eignung für Echtzeit-Anwendungen und zeigt Nachteile sowie Verbesserungsmöglichkeiten.

Abstract

An ocean that reflects wind conditions, reacts to the terrain and interacts with rigid bodies has great potential to improve believability of virtual worlds. Its simulation and rendering need to be fast enough to meet the real-time requirement in conjunction with many other features. This work employs a multi-band Fast Fourier Transform approach to synthesize a mixed sea ocean surface valid in deep water. A sampling strategy for wave spectra suitable for seamless transitions of sea states is introduced, and recommendations for a complete GPU implementation are made. Deep water is coupled with a shallow water simulation to supplement terrain awareness and rigid body interactions. For this, a recent improvement of the Lattice Boltzmann Method with reduced memory usage is adopted and implemented on the GPU. It is found how its properties can be used to cover a large-scale area with varying level of detail which may be of interest beyond the computer graphics research. A performance analysis and discussion of the results demonstrate the suitability for real-time applications, point out deficiencies and suggest exciting future work.

Contents

1 Introduction **1**
1.1 Structure of the Work 2

2 Related work **3**
2.1 Ocean Simulation in Computer Graphics 3
2.2 Deep Water Simulation 4
2.3 Shallow Water Simulation 6
2.4 Choice of Simulation Techniques 9

3 Background **10**
3.1 Oceanographic Fundamentals 10
3.1.1 Waves under Realistic Conditions 12
3.2 Simulating Deep Water 14
3.2.1 Ocean Wave Spectra 14
3.2.2 The Fast Fourier Transform Approach 17
3.2.3 Small Scale Details and Growing Complexity 18
3.2.4 Multi-Band Enhancement 19
3.2.5 Multi-Spectra for Mixed Sea 19
3.2.6 Generating Surface Normals 20
3.2.7 Horizontal Surface Displacement 21
3.3 Simulating Shallow Water 22
3.3.1 Lattice Boltzmann Method for Shallow Water Equations 24
3.3.2 Macroscopic Lattice Boltzmann method for SWE . . . 28
3.4 Surface Mesh 30
3.4.1 Algorithm of the Projected Grid 30
3.4.2 Problems of Projected Grid Approach 32
3.4.3 Possible Improvements 33

4 Implementation **34**
4.1 Simulating Deep Water 36
4.1.1 Controllability 36
4.1.2 Sampling Wave Spectra 37
4.1.3 GPU implementation 41

4.2 Simulating Shallow Water 43
4.2.1 Parameter Clarification 43
4.2.2 Implications of the Parametrization 45
4.2.3 Level of Detail Scheme 46
4.2.4 GPU implementation 50
4.3 Coupling Deep and Shallow Water 53
4.4 Rigid Body Interactions 55
4.5 Output Combination 58
4.5.1 Projected Grid World Positions 59
4.5.2 Combining Shallow Water Detail Layers 60
4.5.3 Combining Deep Water Bands 62
4.5.4 Foam Generation 62
4.5.5 Shoreline Mesh Correction 63

5 Results and discussion **65**
5.1 Deep Water Simulation Results 65
5.1.1 Performance Analysis 65
5.2 Shallow Water Simulation Results 69
5.2.1 Comparison of Simulation Methods 71
5.2.2 Performance Analysis 73
5.2.3 Adaptive Level of Detail 76
5.3 Interaction with Rigid Bodies 77
5.4 Projected Grid Approach 79

6 Conclusion **80**

Glossary **A**

Index of abbreviations **B**

List of Symbols **C**

Bibliography **F**

1 Introduction

Oceans are what makes our planet blue, they are the cradle of life, can be soothing and mesmerizing and at the same time be powerful and destructive. Long before computer graphics existed, oceans, waves and water were subject of research. This accumulated knowledge finds its use in weather forecasting, climate research, maritime simulations, structural analysis, the film industry and many more. In this work ocean rendering and simulation is examined for the usage in a video game, one application area of computer graphics with very strict performance demands. Achieving realistic water behaviour at a large scale with just a few milliseconds of computing time is not an easy task. However, the possibility of doing this has been pushed since the early days of computer graphics and every new hardware generation allow new steps towards the goal.

While oceans have a wide range, the focus shall be on its surface viewed from above with a visible range of several kilometres down to a few meters. This includes both offshore zones with deep water and shore zones with shallow water reacting to the seabed. A distinction of these zones is practical since waves, which are perceived on the sea surface, behave differently if they are affected mainly by gravity or instead confined by the solid sea bed.

Different wave spectra, that were developed using measurements of real-life sea states, are the employed foundation to describe the ocean surface in deep water. The *Fast Fourier Transform* is one way to use these spectra and is capable to consider a great amount of waves for a wide range of wavelengths. Both its application in real-time and seamless transitions of sea states for different wind conditions require special attention. Hereof suggestions and considerations towards improved performance and better controllability are made.

Interactive behaviour of water can be introduced with a fluid simulation that can be particularly performance demanding but also particularly rewarding with its results. The associated branch of research has led to a vast amount of methods suitable for different kinds of applications. Sacrificing or approximating fluid details allow making a fluid simulation feasible for real-time usage. This work adopts a recent improvement of the Lattice Boltzmann Method to solve the shallow water equations and researches how to span large areas while maintaining high details in vicinity to the camera. It is supplemented by interactions between rigid bodies and the simulated ocean surface.

Both deep and shallow water models are combined to deliver the necessary inputs for the ocean to be rendered by an existing rendering solution. The achieved results are discussed to both provide insights into their possibilities and limits and suggest ways for further improvement and future work.

1.1 Structure of the Work

Chapter 2 starts with a computer graphics related overview of ocean simulation. It continues with notable related work classified in oceanographic models to simulate deep and physically-based models to simulate shallow water. This classification largely is used throughout the work to enable reading the respective topics specifically. In section 2.4 requirements are presented and the choice of simulation technique is justified.

Essential background knowledge is introduced in section 3.1 of chapter 3. A detailed introduction of wave spectra, the basic approach of synthesizing the ocean surface in deep water with the Fast Fourier Transform (FFT) and more recent improvements follows in section 3.2. Shallow water simulation with the Lattice Boltzmann method (LBM) and its recent enhancement are presented in section 3.3. The last part 3.4 of the chapter introduces and discusses an approach to generate the ocean's surface mesh.

Chapter 4 covers this work's implementation, proposed ideas and suggestions for computations on the graphics processing unit (GPU). An initial overview is followed by the Fourier Transform based deep water model in section 4.1. Here, a wave spectra sampling strategy is recommended and its GPU computation is presented. Section 4.2 first clarifies the employed fluid simulation's properties and continues to derive a level of detail (LOD) system based on the gained insights. A focus is put on stability conditions and the GPU implementation. Coupling of deep water with the shallow water simulation in section 4.3, rigid body interactions 4.4 and the combination and utilization of simulation outputs in section 4.5 close the chapter.

Results are discussed in chapter 5 giving an assessment of what worked or should be improved. This is supported by the comparison with existing work and an in-depth performance analysis. The conclusion in chapter 6 sums up the results and provides directions for future work.

2 Related work

This introduction of related work starts with the review of surveys providing an overview and means of classification which are useful in the course of the work. It is followed by briefly summarized notable examples which are considered to serve as a good starting point for further research.

2.1 Ocean Simulation in Computer Graphics

Darles et al. [Dar+11] present ocean simulation techniques classified into two main categories. The first category includes parametric or spectral models based on oceanographic research and the second physically-based models from the computational fluid dynamics research branch. Former make use of theoretical and empirical results of the oceanography research field that are especially used and suited to describe the properties of the sea and its surface in deep water. Shallow water, whose complex motion is affected and defined by the bottom of the sea, is best modelled with fluid dynamics where Navier-Stokes equations are emphasized in particular. After this introductory observation, Darles et al. stick with the classification of ocean simulation into deep and shallow water and provide further subdivisions. Thus, deep water simulations can be divided in spatial domain approaches, Fourier domain approaches and in hybrid approaches that combine both previous ones. Shallow water simulations are divided in Eulerian approaches usually solving the Navier-Stokes or related equations on a 2D or 3D grid, in Lagrangian approaches that employ particles instead of grids and again in hybrid approaches.

Manteaux et al. [Man+17] focus on adaptive physically-based models for various simulations including fluids and present a taxonomy with temporal and geometric adaptivity as the two main categories. Their definition of adaptivity requires a model or simulation method to adapt itself at runtime based on the simulation state in order to meet certain criteria like reducing computational complexity or improving quality in specific areas. Temporal adaptivity includes selecting an appropriate time step either globally for the entire simulation domain or locally, both under consideration of some criteria that usually are required for accuracy and stability reasons. It is mentioned that changing the time step at runtime highly depends on the integration scheme, as some rely on fixed time steps. Additionally, replacing an integration scheme or freezing time under certain conditions are considered to be temporal adaptive solutions. Geometric or spatial adaptive techniques can utilize a refinement criterion to determine where a refinement scheme should to be applied. Simple refinement criteria may be the distance to or curvature of a surface. The depth of a liquid or its surface detail measured by a deformation factor are examples for more sophisticated criteria. Examples for refinement schemes are spatial structures like quad-, or octrees, mesh refinement,locally increasing the amount of samples or multi-scale methods that hierarchically couple different resolutions. Instead of refining, moving grid methods rely on placing simple uniform grids at point of interests like specific objects in a scene, and moving the grid with them. While this keeps the simple nature of uniform grids, the coupling of overlapping grids requires special attention. Regarding fluid simulations so-called mixed models combining 2D and 3D representations or Eulerian and Lagrangian approaches are highlighted. Here, the seamless coupling is noted to be a difficulty. Finally, Manteaux et al. point out that adaptive methods are not free of limitations as they are difficult to develop and tend to increase the complexity of the model, which can have a negative impact on GPU implementations.

Kellomäki [Kel17] considers the topic of water simulation in terms of its application in computer games. He points out that performance requirements are much stricter than they are for research purposes because a game cannot use all available system resources just for the water simulation. Furthermore, the robustness of the simulation is important, as instabilities or artefacts may ruin the user experience. Instead of reaching for a more stable simulation by sacrificing performance, Kellomäki suggests that it is preferable to accept a less realistic simulation result and rather aim for plausibility.

2.2 Deep Water Simulation

In 1986, Fournier and Reeves [FR86] describe a model to parametrically describe the surface of ocean waves using modified *Gerstner Waves* which date back to the 19th century. This is most likely the first introduction of Gerstner Waves to the computer graphics research field. Cutting the surface along a vertical coordinate plane exposes that the basic model is a trochoid. Thus, a single Gerstner Wave consists of two sine waves on the surface plane. Their modifications and introduction of parameters allow to manipulate the curves steepness and shape, to align the waves with the wind and lining them up with the beach’s slope. Evaluating the surface’s curvature and the speed of a point on the surface allows the detection of breaking waves and the generation of spray and foam.

Bruneton, Neyret, and Holzschuch [BNH10] model deep ocean waves with a sum of 60 Gerstner Waves and choose the controlling parameters according to the Pierson-Moskowitz spectrum. They focus on a hierarchical representation of the waves to seamlessly fade between geometry, surface normals and an ocean surface BRDF depending on the distance to the camera.

Bucci and Longchamps [BL17] present the ocean technology used for the video game *Hitman* (2016) at the Game Developers Conference (GDC) 2017 which also relies on Gerstner Waves. In addition to a base layer of 8 Gerstner Waves travelling in different directions for open water, they model coastal waves that match the curvature of the shoreline. Placement of coastal waves is done manually and the user controls their look and spatial extent with parameters including the direction and the curvature. They justify their decision for Gerstner Waves over waves created with the FFT foremost with the computational efficiency, the simplicity to generate surface normals and the less visible spatial tiling.

Tessendorf [Tes01] uses Gerstner Waves mainly as a way to lead up to the topic of Fourier Transform, as they share some fundamental similarities. Instead of adding up multiple Gerstner Waves in spatial domain, he recommends using the FFT which is superior in evaluating sums. The result is a single tileable heightmap of the ocean surface of adjustable spatial size and resolution. Besides a heightmap surface normals can either be computed with central differences or by means of the slope obtained through Fast Fourier Transform. Sharper wave crests can be obtained with an additional displacement to achieve the look of rough sea. At the time of 2001 his approach was already used in cinema for *Waterworld* (1995) or *Titanic* (1997) but since then it was adopted and improved multiple times and can clearly be considered a groundbreaking work. The main drawback of the approach is that the Fourier grid's resolution is directly connected to the surface's level of detail. In real-time applications it is not possible to use resolutions matching those of films and thus the resulting surface lacks high frequency waves.

Fréchet [Fré06] proposes an adaptive sampling scheme of the underlying wave spectrum to concentrate on the interval with the highest energy. Additionally, for close up views of the surface, short waves are preferred by taking more samples in low frequency intervals. This approach enables interactive frame rates back in 2006 with a richer surface than Tessendorf's approach.

As LeBlanc et al. [LeB+12] point out, non uniformly spectral sampling loses the computational savings of the FFT. They present a different approach to tackle the problem of a lack in details, by splitting the spectrum in several narrow bands, each of which is individually transformed using the FFT. With four separate Fourier grids of low resolution they obtain visually comparable

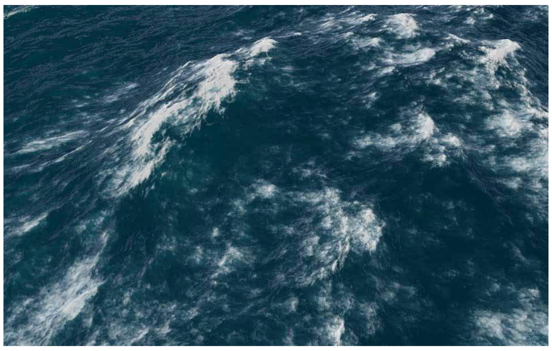


Figure 2.1: Image of an ocean in stormy conditions from [Tch15] shows convincing foam on wave crests.

results to a single high resolution Fourier grid as used in the film *Titanic* (1997) whilst achieving an immensely higher frame rate.

The GDC presentation from Tcheblokov [Tch15] shows a similar technique from *NVIDIA*'s *WaveWorks* as used in the game *War Thunder* (2015). Here, the frequency bands are called cascades and it is suggested to fade out those cascades containing higher frequencies at greater distance. He shows improvements alongside specific recommendations for foam simulation (which can be seen in figure 2.1) based on Tessendorf's work and shore interactions that has similarities to Fournier and Reeves' parametric shore waves.

At GDC 2019 Mihelich and Tcheblokov [MT19] show a more recent version of *NVIDIA*'s *WaveWorks* adopting an advanced wave spectrum as input for a multi-band Fourier Transform approach that allows modelling waves emerging from local wind and distant storms. They also greatly improve the visual quality of surface shading due to the usage of a bidirectional reflectance distribution function (BRDF) for surface shading and subsurface light scattering.

Park and Park [PP20] propose a mixed sea simulation for use in real-time maritime simulators. They employ a multi-band approach with the addition of multiple sea systems modelled by different sea wave spectra. In a single sea system all waves travel in the same direction as they are synthesized using the same input parameters. Multiple sea systems with appropriate wave spectra enable modelling local wind sea and swells, which correlates to Mihelich and Tcheblokov's presentation of *NVIDIA*'s *WaveWorks*.

2.3 Shallow Water Simulation

As shown by Darles et al., physically-based water simulation can be coarsely divided into Lagrangian and Eulerian approaches. The research for this work has revealed, that Lagrangian approaches of water simulation usually have a very restricted spatial simulation domain as a large domain like an open ocean requires an enormous amount of particles. Therefore, their usage in large scale real-time simulations is either supplementary in hybrid approaches for local improvements of a coarser grid-based approach, or the term real-time is less strictly defined and does not fulfil the performance requirement for video games.

Opting for height field based 2D grids simplifies the simulation of fluid enough to become viable for large scale usage in video games. Examples supporting this are the games *From Dust* (2011) and *Cities: Skylines* (2015) that use the simulation for game-play elements [Kel17]. An important consideration is that all 2D height field based fluid simulations have the common drawback of missing important features due to the reduction of dimensionality. Among these are waterfalls, splashes and breaking waves. Nevertheless, the reduced computational cost is what still makes them attractive.

A crucial aspect that always has to be considered is the stability of the simulation. While numerical integration schemes may have different stability conditions, they are always connected to the time step, the spatial extent of the simulation domain and the speed at which information propagates [Man+17].

One widely adopted grid-based approach for 2D fluid simulation is the virtual pipe model proposed by O'Brien and Hodgins [OH95]. Vertical columns of water are connected by virtual pipes to its four or eight neighbours through which the flow of water is simulated. Water depth is explicitly forward integrated based on the in and outflow to obtain the water depths on a grid. This model was picked up almost simultaneously but independently by Maes, Fujimoto, and Chiba [MFC06] and Mei, Decaudin, and Hu [MDH07]. Both describe a possible GPU implementation, which is an important property for a simulation to be of use in a large-scale real-time application. Kellomäki adopts Mei, Decaudin, and Hu's solution and evaluates its suitability for computer games. The video game *Sea of Thieves* (2017) also uses their approach [Ang+18] alongside some extensions to supplement missing features of waterfalls and streams. Since it is an explicit method, the time step must be small enough for the information to propagate within the limits of the lattice size [Man+17; MDH07]. The exact limit on the time step is affected by various factors and thus can only be approximated [MDH07]. Therefore, it may be found experimentally depending on the simulated problem as done by Kellomäki using a time step of 25 ms. Additionally, no advection of the velocity like in the shallow water equations is performed which does not allow for vortices to appear [Kel17]. A velocity field can be reconstructed but the velocity itself has no effect on the fluid simulation.

Besides the pipe model there is a large array of works that refer to the shallow water equations (SWE) that are derived from the Navier-Stokes

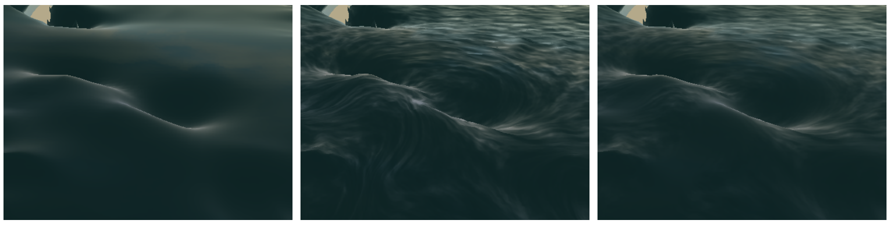


Figure 2.2: The shallow water simulation from Chentanez and Müller renders vortices visible with the addition of high frequency FFT waves and texture coordinates that are advected with the velocity field [CM10].

equations. With several assumptions and simplifications the SWE allow to simulate the water's depth and horizontal velocity, however they cannot correctly handle deep water, which is already suggested by their name.

Thurey et al. [Thu+07] utilize them to simulate a water height field and detect steep wave fronts at which they modify the mesh to reintroduce the missing feature of breaking waves. The velocity is used to spawn additional particles depicting spray.

Chentanez and Müller [CM10] solve the SWE using an explicit integration scheme and handle potential sources of instabilities with special care. Enforcing certain limits on the depth and velocity is their main idea to keep the integration stable. Unfortunately their proposed stability enhancement do not guarantee stability. In addition to the grid-based simulation they employ a particle simulation to spawn both spray and foam near breaking waves or waterfalls and splashes from interactions with rigid bodies. For the stable simulation with a reasonable time-step the lattice size gets rather high. Therefore, they enhance the surface with high frequency details by sampling waves created by a FFT and advecting the associated texture coordinates with the velocity field (see in figure 2.2).

A different approach of solving the SWE is based on the LBM which experienced an increase of interest in the last decades. It differs from numerical solutions as it based on statistical physics and simulates the flow of a fluid in terms of fictive microscopic particles moving along a set of discrete lattice vectors [Zho04, p. 19; Tub10]. The microscopic formulation allows simple and efficient computations and can correctly describe different macroscopic governing equations, including the SWE [Zho04, p. 19; Tub10]. Besides numerous mathematical research publications from Zhou who actively researches this field with a focus on SWE [Zho04; Zho11; ZL13; Zho19b], the dissertation [Oje13b] and published paper [Oje13a] presents the LBM in a computer graphics context. Ojeda couples a 2D shallow water LBM simulation with a particle simulation to add spray at wave fronts. What makes the LBM appealing is, that it has well-researched stability conditions and it is a suitable candidate for GPU implementation because computations only operate in a local neighbour.

Before closing this brief summary of related work it shall be mentioned that there is an abundance of excellent and promising literature about simulating and rendering oceans let alone simulating fluids. Listed works were selected because they met certain criteria. They were either considered useful to provide an overview and insight into the course of research, or found to be promising due to introducing novel ideas, identifying related problems, proposing possible solutions or achieving desirable results.

2.4 Choice of Simulation Techniques

An initial informal gathering of requirements with *Limbic Entertainment* provides the basis of decision-making. Regarding performance a frame budget of approximately 2 ms was endorsed which aligns itself with the statement of Kellomäki in [Kel17] about a strict performance requirement for computer games. As a functional requirement, the simulation needs to support interactions with its environment. In particular waves need to be aware of the seabed as they are able to reach the shore, be affected by wind in a plausible way and preferably they should not run up the shores of an island uniformly from all directions. There needs to be some interaction between the water and rigid bodies. Examples for rigid bodies are anchored, sailing or sinking ships and debris hitting the water surface. Possible but non-mandatory effects on the water surface are foam, splashes and spray whilst ships should naturally be able to float. Other non-mandatory ideas include the capability to see the terrain or sunken ships below the water surface, the support of a day and night cycle or a maelstrom. Finally, the primary view is a top-down perspective with the option for occasional close-up views.

With these requirements in mind, a combination of the FFT for simulating deep water with the LBM for simulating shallow water was chosen. The FFT approach promises the simulation of a highly realistic ocean surface with the possibility to model varying wind conditions. It already met the requirements of the film industry and there has been ongoing research for usage in real-time applications. The required rigid body and shore interactions can be achieved with a physically-based shallow water simulation. Here, the LBM is promising due to its simple computations and possible GPU implementation. In contrast to other numerical solutions like the explicit integration scheme as proposed by Chentanez and Müller in [CM10], the LBM for the SWE makes clear statements on stability conditions. Moreover, a recent publication of Zhou proposes an updated concept of the LBM which sparks interest, since Zhou calls it a revolutionary and precise minimal method [Zho19b]. It removes the drawback of a high memory requirement and allows direct use of water depth and velocity. Since this are major improvements for the practical usage in real-time application this new formulation of the LBM method was chosen for the context of this work.

3 Background

Henceforth, the classification of water in the two zones of deep and shallow water is adopted and it is shown to be well-founded. Different physical properties in these zones allow for approximations to be made which in turn render some approaches useful and others unsuitable. In this chapter, after an introduction of necessary basics is given, the chosen approaches to model and animate an ocean's surface in shallow water and in deep water are shown in more detail.

3.1 Oceanographic Fundamentals

The surface of water is rarely flat and still but rather in motion. Whether waves are caused by wind, gravity, tectonic movements or impacting bodies they share two common properties. Energy is propagated and the displacement is travelling through the medium without permanently displacing it as a whole [Bee97, p. 59]. Before introducing specific approaches of water simulation, fundamental nomenclature and their meaning shall be clarified. The following introduction is based on [Bee97, p. 59−63], if not stated otherwise.

Idealized waves like sinusoidal waves or the waves depicted in figure 3.1 are periodic and work well to describe wave parameters. Crests and troughs propagate at a phase speed c. Measuring the time it takes for two successive crests or any other corresponding position passing a fixed point yields the period T and the frequency f as the period's inverse. The distance between such successive points is the wave length λ. The height of a wave is the difference between the lowest and highest point of the wave whereas the amplitude is the difference between the mean sea level and the crest or trough. Finally, the water depth d is the distance from the bed to the mean sea level.

Tracking the position of a particle in water reveals a circular motion which is a visual explanation of why pi is omnipresent in wave theory. Therefore, using the definition of the angular wave frequency $\omega = 2\pi f$ is convenient. In terms of frequency and period, the phase speed (or celerity) is defined as $c = f/T$. Using angular units it can be equivalently written as $c = \omega/k$.

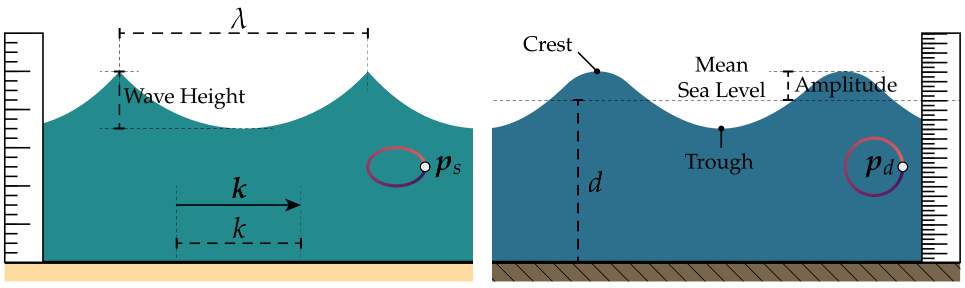


Figure 3.1: A schematic illustration of shallow water (left) and deep water (right) presents basic waves parameters based on [Bee97, pp. 60–62]. The trajectory of a particle in shallow water $\boldsymbol{p}_s$ in comparison to that of a particle in deep water $\boldsymbol{p}_d$ visualizes the idea of shallow water feeling the sea bed.

Here, k is the angular wave number which is short for $(2\pi)/f$. Observing the motion of a wave in an ideal and closed environment easily reveals it is moving in a direction. The wave vector $\boldsymbol{k}$ points into this direction of wave propagation and its magnitude equals the wave number k [Tes01].

If the water depth is high in relation to the wavelength, the resisting influence of the ocean bed on the surface wave is negligible. In shallow water, on the other hand, the wave is heavily influenced by the ocean bed. The particle trajectories in figure 3.1 illustrate this effect and show an elliptical motion in shallow water and a circular motion in deep water. Put into numbers, water is considered deep with a depth greater than $1/4$ of the wavelength and water is considered shallow with a depth smaller than $1/20$ of its wavelength. The name *shallow water* can be misleading at first since it is not a classification in terms of an absolute water depth. However, the conditions for shallow water are usually satisfied in water of little depth. The exact ratios for classification may vary depending on literature.

Angular wave frequency ω and the wave number k are connected by the so-called dispersion relation, which is why sometimes the angular wave frequency is written as a function of the wave number $\omega(k)$. This important relation describes the property of waves with different wave lengths moving at different speeds [Tes01] and constitutes a limiting factor on the wave speed c. Distinguishing shallow and deep water allows for a mathematical approximation in these zones, whereas the intermediate zone would require a more sophisticated formula [Ste08, pp. 274–275].

$$c = \frac{\lambda}{T} = \frac{\omega}{k} = \begin{cases} \sqrt{gk} & \text{if } d > \lambda/4 \\ \sqrt{gd} & \text{if } d < \lambda/20 \\ \sqrt{gk \tanh(kd)} & \text{in between} \end{cases} \tag{3.1}$$

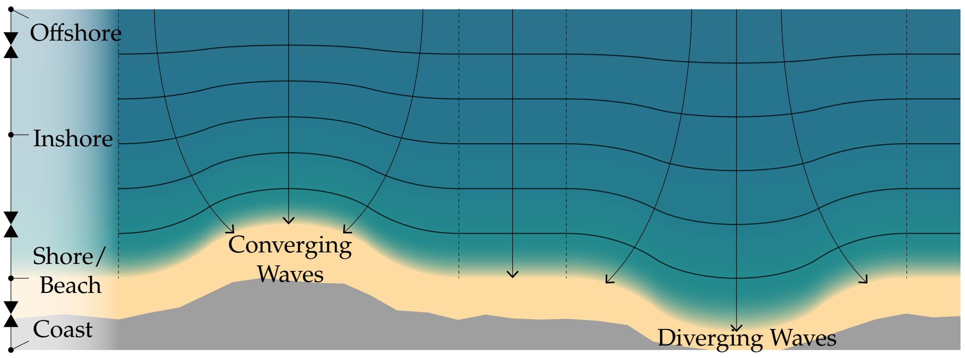


Figure 3.2: A schematic illustration how waves in shallow water near the shore heavily are influenced by the seabed topology. On the left side of the figure the different zones offshore, inshore, shore and coast are labelled. This illustration is based on [Bee97, pp. 30–32].

3.1.1 Waves under Realistic Conditions

Under realistic conditions waves on the ocean are not as ideal as depicted in figure 3.1 and there is not just a single wave of a specific wavelength. To make a statement on the height of waves on the ocean, either the average of the significant wave height can be used. The significant wave height is the average of the highest third of waves height which approximately correlates to what an observer estimates the height to be [Bee97, p. 72]. Surface waves can be classified by their period or frequency. For an observer it becomes particularly difficult to estimate the height of waves Capillary waves are very short waves of high frequencies and are caused for example by surface tension. Tides form the other extreme with very long waves and periods of several hours. In between are the waves caused by wind, with swells in the lower frequency range and local wind waves covering a wider spectrum. As the name suggests, local wind waves are created by local wind whereas swells originate from wind in areas further away. In strong local wind conditions waves of rather low frequencies cause a choppy surface. But during a windless day the water surface is not necessarily free of waves as for instance coastal water in California can exhibit swells from distant storms near New Zealand [Bee97, p. 59]. This work uses the term of a wave system or sea system to describe a group of waves that shares characteristics under which they were created.

When deep water waves reach the shallow water near the shore, they undergo changes due to the speed being depth limited as expressed in equation 3.1. As they slow down the wavelength decreases and successive crests and troughs are getting closer. Non-parallel arriving waves will always bend parallel to the shore which is known as refraction, because a point of the wave front closer to the shore slows down earlier than a more distant point of the wave. Different seabed topologies cause different refraction patterns as illustrated in figure 3.2. In a bay, waves are diverging and as a result the wave heights are lower. A headland experiences exactly the opposite, where the waves are converging towards the tip of the headland resulting in much higher waves.

As the speed c in shallow water is directly related to the water depth, the wave's speed at crests and troughs differs significantly upon approach the shoreline. Crests moving at higher speeds will eventually overtake the slower moving troughs and the wave will break. Depending on the beach's slope and the ratio of wave height to wave length of the arriving deep water wave, different types of breaking waves occur.

For a more in-depth introduction into oceanography the books *Environmental Oceanography* [Bee97] by Beer and *Introduction to Physical Oceanography* [Ste08] by Stewart are a possible starting point.

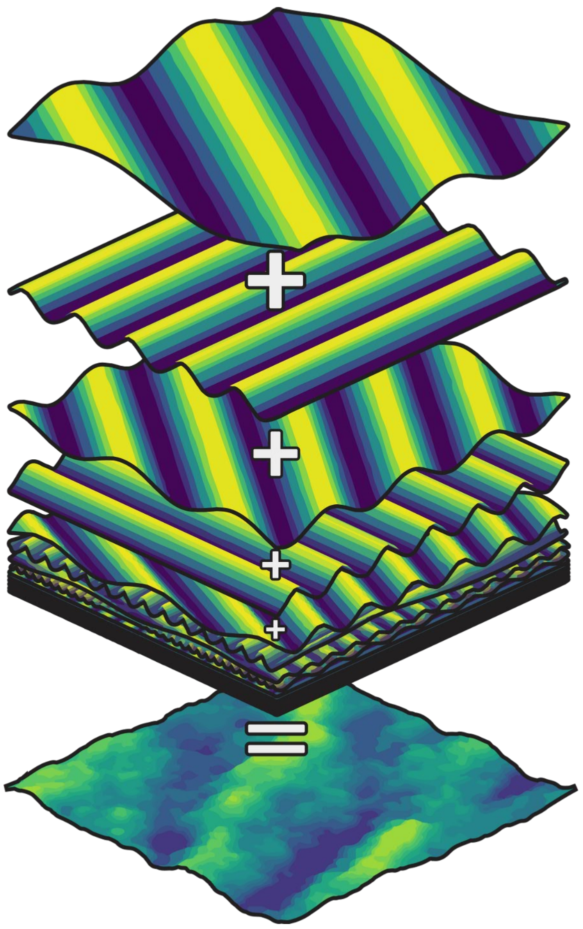


Figure 3.3: The ocean surface can be synthesized by adding up many sinusoidal waves. This schematic illustration is based on the idea of *Practical Methods for Observing and Forecasting Ocean Waves by Means of Wave Spectra and Statistics* [PJN55].

3.2 Simulating Deep Water

Swells from distant storms around the globe and local wind waves build a sea with waves of highly diverse frequencies travelling in different directions. To analyse the ocean's wave components or to synthetically reconstruct it, utilizing the French mathematician Jean-Baptiste Joseph Fourier's (1768-1830) insights proves useful. He discovered that any periodic wave can be expressed as a sum of an infinite number of sine waves [Bee97, p. 70]. Figure 3.3 depicts this idea transferred to the surface the ocean.

As mentioned previously, it is possible to model waves in spatial domain adequately with Gerstner Waves or other periodical functions. While advantages of working in the spatial domain are comprehensibility and the computational upside of directly calculating the sought variables, the main disadvantage is that adding up signals in spatial domain is not the fastest option. The Fast Fourier Transform, on the other hand, is very efficient in quickly adding up signals. In fact, it is so efficient that a proper approach to generate signals is required. The next section introduces the concept of wave spectra which enables exactly that.

3.2.1 Ocean Wave Spectra

A nice way to introduce wave spectra may be to look at the steps Park and Park use in their work [PP20] to verify the physical validity of their result. First, they synthesize an ocean height field using an extended version of Tessendorf's FFT approach. This involves sampling spectral densities of a wave spectrum which are required as input for the transformation. They proceed by recording the wave heights at a single location on the ocean surface for a certain period of time (see figure 3.4). This signal in time-domain can be transformed back to frequency-domain by applying the Fourier Transform and Park and Park are able to reconstruct the original wave spectrum quite accurately. Details on how to calculate a wave spectrum are not considered in this work but can be found in chapter 16 of *Introduction to Physical Oceanography* [Ste08].

Instead of measuring the heights of a synthesized ocean, surface spectra used in oceanography research obtain measurements from buoys or ships. More specifically, the Pierson-Moskowitz Spectrum uses measure-

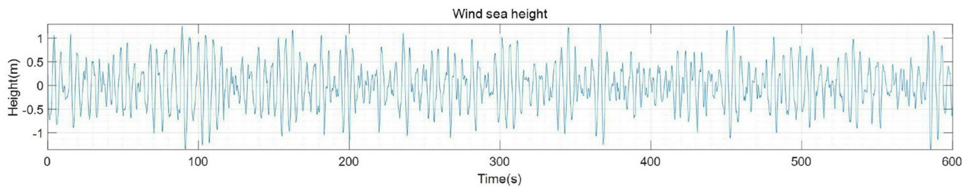


Figure 3.4: A time line of recorded wave heights measured in a wind sea by [PP20].

ments taken on two weather ships in the North Atlantic [PM63] during different wind conditions in 1963, whereas the Joint North Sea Wave Project (JONSWAP) obtained data from thirteen wave stations in the North Sea in 1969 [HO73]. The obtained spectra were used to find a function of frequency that both closely matches the recorded data and can be manipulated by several convenient parameters [Ste08].

The Pierson-Moskowitz spectrum

The Pierson-Moskowitz Spectrum models a fully developed sea which assumes that waves come into a state of equilibrium after the wind blows steadily for a long time covering a large area [Ste08].

$$\mathrm{S}(\omega) = \frac{\alpha g^2}{\omega^5} \exp\left(-\beta\left(\frac{\omega_p}{\omega}\right)^4\right) \tag{3.2}$$

Here, $\alpha = 0.00779, \beta = 0.74$ are constants controlling intensity and shape and $\omega_p = g/(U_{19.5})$ is the peak frequency with the wind speed $U_{19.5}$ being measured at a height of 19.5 m above the sea surface [PM63].

The JONSWAP spectrum

The data of the Joint North Sea Wave Project indicated that in reality, a sea is not fully developed but rather continues to stay in a developing state [HO73]. They include this property in their spectrum function with a wind fetch F parameter, which allows to control the distance the wind has travelled over the sea. The γ parameter allows to adjust the peak of the spectrum and enhances interactions between waves, which turns out to be visually important [Ste08].

$$\mathrm{S}(\omega) = \frac{\alpha g^2}{\omega^5} \exp\left(-\frac{5}{4}\left(\frac{\omega_p}{\omega}\right)^4\right)\gamma^r \tag{3.3a}$$

$$r = \exp\left(-\frac{(\omega-\omega_p)^2}{2\sigma^2{\omega_p}^2}\right) \tag{3.3b}$$

Here, $\alpha = 0.00776\left((U_{10}^2)/(Fg)\right)^{0.22}$, the peak frequency $\omega_p = 22\left(g^2/(U_{10}F)\right)^{1/3}$, the peak enhancement factor γ and $\sigma = 0.07$ for $\omega \leq \omega_p$ and $\sigma = 0.09$ for $\omega > \omega_p$ [HO73; Ste08].

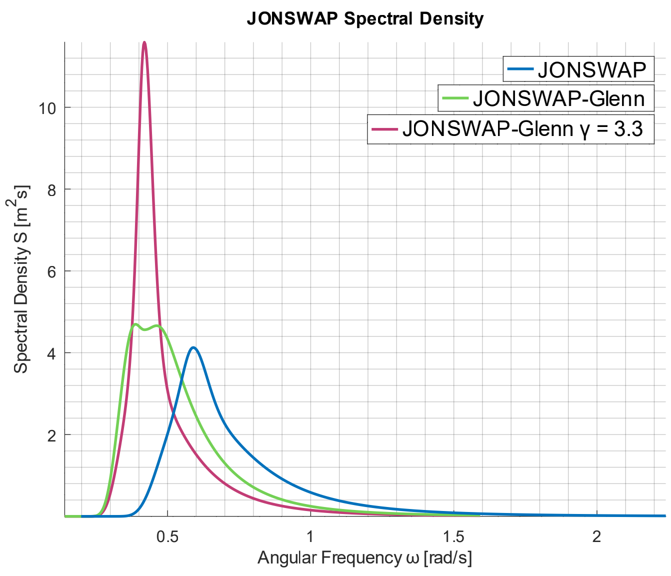


Figure 3.5: Comparison of JONSWAP and JONSWAP-Glenn Spectrum. Parameters for JONSWAP: $F = 500\,\text{km}, U = 10\,\text{m}\,\text{s}^{-1}, \gamma = 1.51$ and for JONSWAP-Glenn: $T_p = 15\,\text{s}, H_s = 2\,\text{m}$. Two plots of JONSWAP-Glenn highlight the impact of the peak enhancement factor γ as suggested in [PP20] compared to $\gamma = 3.3$.

The JONSWAP-Glenn spectrum

The JONSWAP-Glenn spectrum [Ola+13] is a variation of the JONSWAP spectrum which is suitable for modelling the low frequency swells [PP20].

$$S(f) = c\left(\frac{f}{f_p}\right)^{-5} \exp\left(-\frac{5}{4}\left(\frac{f_p}{f}\right)^4\right)\gamma^r \tag{3.4a}$$

$$c = \frac{5{H_s}^2}{16 f_p}\left(1.15 + 0.1688\gamma - \frac{0.925}{(1.909+\gamma)}\right)^{-1} \tag{3.4b}$$

H_s describes the significant wave height which is the average height of the waves in the highest third of the wave spectrum [PP20]. The peak frequency f_p can be expressed by means of the peak period T_p. Park and Park suggest choosing the peak enhancement factor γ in relation to H_s and f_p with $\gamma = 9.5{H_s}^{0.34} f_p$. Its effect on the spectrum can be seen in figure 3.5.

Applying directional spread

In addition to wave spectra, the direction of travel can be taken into account since they tend to travel in the direction of wind. This can be applied in frequency domain with a spreading function published in [LCS63] and

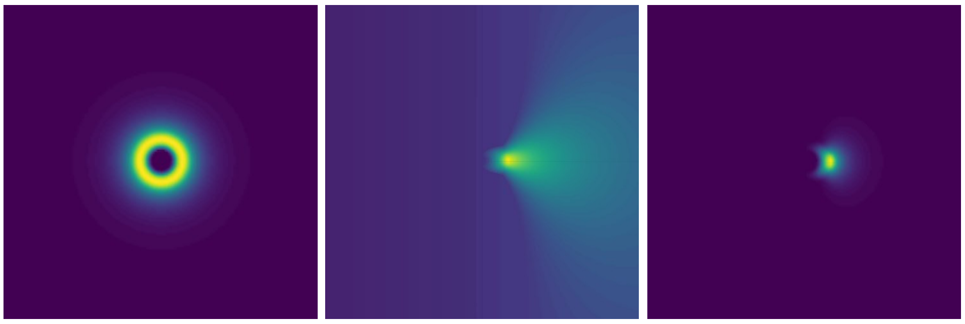


Figure 3.6: The JONSWAP spectrum (left) multiplied with the directional spreading (center) yields the directional wave spectrum (right).

obtained from [PP20]:

$$\mathrm{Dir}\left(\omega,\theta\right) = \frac{\Gamma\left(\mathrm{s}(\omega)+1\right)}{2\sqrt{\pi}\Gamma\left(\mathrm{s}(\omega)+0.5\right)}\left(\cos\left(\frac{\theta}{2}\right)\right)^{2\,\mathrm{s}(\omega)} \tag{3.5a}$$

$$\mathrm{s}(\omega) = 16.0\left(\frac{\omega}{\omega_p}\right)^{\mu} \tag{3.5b}$$

$$\mu = \begin{cases} 5.0 & \text{if } \omega \le \omega_p \\ -2.5 & \text{if } \omega > \omega_p \end{cases} \tag{3.5c}$$

with the gamma function $\Gamma = (n-1)!$, θ as the angle between the wind and wave travel direction and s as a spread parameter.

Figure 3.6 shows the combination of a selected wave spectrum $\mathrm{S}(\omega)$ and the directional spreading function $\mathrm{Dir}\left(\omega,\theta\right)$ which returns the directional wave spectrum [PP20]:

$$\mathrm{S}(\omega,\theta) = \mathrm{S}(\omega)\,\mathrm{Dir}\left(\omega,\theta\right) \tag{3.6}$$

3.2.2 The Fast Fourier Transform Approach

In [Tes01] Tessendorf describes how to synthesize a tileable surface patch of ocean from a wave spectrum with the inverse FFT. Thus, to generate the ocean surface in form of a square height field $h(\boldsymbol{x},t)$ at a position $\boldsymbol{x} \in \mathbb{R}^2$ and an absolute time t a finite number of wave components sampled in the Fourier domain are added:

$$h(\boldsymbol{x},t) = \sum_{\boldsymbol{k}} \tilde{h}(\boldsymbol{k},t)\exp(i\boldsymbol{k}\cdot\boldsymbol{x}) \tag{3.7}$$

For the height field to be a $N \times N$ grid of side length L, the wave vector $\boldsymbol{k} = \left(k_x, k_y\right)$ also lies on a square grid, such that:

$$k_x = \frac{2\pi j_x}{L} \text{ with } \frac{-N}{2} \leq j_x < \frac{N}{2} \tag{3.8a}$$

$$k_y = \frac{2\pi j_y}{L} \text{ with } \frac{-N}{2} \leq j_y < \frac{N}{2} \tag{3.8b}$$

At time t, the Fourier amplitudes $\tilde{h}(\boldsymbol{k}, t)$ are

$$\begin{aligned}\tilde{h}(\boldsymbol{k}, t) = {} & \tilde{h}_0(\boldsymbol{k}) \exp(i\omega(k)t) \\ & + \tilde{h}_0^*(-\boldsymbol{k}) \exp(-i\omega(k)t)\end{aligned} \tag{3.9a}$$

$$\tilde{h}_0^*(\boldsymbol{k}) = \tilde{h}(-\boldsymbol{k}, t) \tag{3.9b}$$

The given formulation of equation 3.9a allows using real instead of complex numbers, therefore preserving the complex conjugation property in equation 3.9b and yields a seamless tileable spatial patch [Tes01; Fré06]. This has the benefit of a reduced computation time by a factor of two [Fré06]. Following [Fré06] and [LeB+12], the Fourier amplitudes $\tilde{h}_0(\boldsymbol{k})$ and in turn the amplitude of a wave component $\mathrm{a}(\boldsymbol{k})$ are:

$$\tilde{h}_0(\boldsymbol{k}) = \mathrm{a}(\boldsymbol{k}) \exp(i\varphi) \tag{3.10}$$

$$\text{where } \varphi \text{ is uniformly random in } (0, 2\pi]$$

$$\mathrm{a}(\boldsymbol{k}) = \sqrt{-\log(r)\, \mathrm{S}\left(\sqrt{gk}, \theta(k)\right) \sqrt{\frac{g}{k^3}} \Delta\boldsymbol{k}} \tag{3.11}$$

$$\text{with } \Delta\boldsymbol{k} = \frac{4\pi^2}{L^2} \text{ and } r \text{ is uniformly random in } (0, 1]$$

Different versions of above equations can be found in literature either due to different variations in random number usage or simply due to mathematical restructuring. Equation 3.11 contains the directional wave spectrum from equation 3.6 where any spectrum can be plugged in. Furthermore, the dispersion relation for deep water from 3.1 is used because the spectrum function's angular wave frequency input ω takes the form of $\sqrt{gk}$.

3.2.3 Small Scale Details and Growing Complexity

A fundamental property of the FFT is that its input needs to be equally spaced which is clearly the case according to previous definition of wave vector $\boldsymbol{k}$. Since the wave vector depends on the grid size N and the spatial length L, the maximum angular frequency to sample the spectrum depends on N and L as well. For a large spatial length, the grid size has to be chosen accordingly to include high frequency waves with low wavelengths.

Increasing N also increases the computational cost, and the complexity for the FFT is $\mathcal{O}(N^2 \log N)$ [LeB+12].

Decreasing the grid size N together with the spatial length L allows the sampling of high frequencies in the spectrum with a low computational cost for the Fast Fourier Transform. However, if the sampling interval is high and the amount of sampled spectral densities is low, the sampled locations are not equally spaced but shifted towards the high frequencies due to the square root of the deep water relation. Finally, a small value for L means the generated ocean tile is small. Since it will be repeated throughout spatial domain, it exhibits visible repetition patterns.

3.2.4 Multi-Band Enhancement

To remedy these issues, LeBlanc et al.s [LeB+12] propose a multi-band approach that consists of multiple low resolution FFTs with each sampling different ranges of the used wave spectrum. This allows covering a wider wave number range at a much lower computational cost and without visible repetition pattern. A number of bands M are evaluated independently and then added in spatial domain:

$$h(\boldsymbol{x}, t) = \sum_{m=1}^{M} \left(\sum_{\boldsymbol{k}_m} \tilde{h}(\boldsymbol{k}_m, t) \exp(i \boldsymbol{k}_m \cdot \boldsymbol{x}) \right) \tag{3.12}$$

with $L_1 > L_2 > \ldots > L_M$ and the banded wave vector $\boldsymbol{k}_m$ made up of the following components

$$k_{mx} = \frac{2\pi j_x}{L_m} \text{ with } \frac{-N}{2} \leq j_x < \frac{N}{2} \tag{3.13}$$

$$k_{my} = \frac{2\pi j_y}{L_m} \text{ with } \frac{-N}{2} \leq j_y < \frac{N}{2} \tag{3.14}$$

To prevent overlapping bands, LeBlanc et al. suggest setting the wave amplitudes a($\boldsymbol{k}_m$) to 0 where $k_m \leq (N\pi)/(L_{m-1})$ [LeB+12].

Their presented evaluation finds that the frame time of updating 4 bands with grid size 64×64 is similar to a single 128×128 grid. Meanwhile, the visual quality exceeds that of a 1024×1024 grid.

3.2.5 Multi-Spectra for Mixed Sea

Park and Park [PP20] use a somewhat similar multi-band approach with the addition of using different directional wave spectra for multiple local wind wave and swell systems. This allows modelling a mixed sea state which is required by Park and Park for the realistic effect on ships in a real-time maritime simulator.

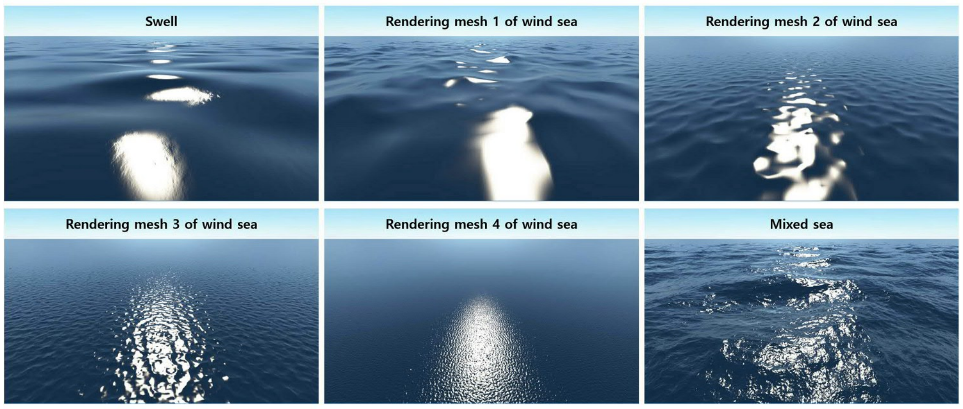

Swell
Rendering mesh 1 of wind sea
Rendering mesh 2 of wind sea
Rendering mesh 3 of wind sea
Rendering mesh 4 of wind sea
Mixed sea

Figure 3.7: This image shows the combination of a single-banded swell sea with a wind sea system split into 4 bands into a mixed sea state [PP20].

The outputs of multiple sea systems, each of which may be divided into a few numbers of bands, are combined in spatial domain (see figure 3.7). For each sea system, a main sampling range is defined around the spectrum's peak frequency where most of the energy is found. Swells are modelled with the JONSWAP-Glenn spectrum and wind sea with the JONSWAP spectrum. To properly match bands to the spatial domain, i.e. the frequencies of the band match the spatial resolution, Park and Park define L as a ratio of N and the band's maximum frequency f_{max}:

$$L = \frac{gN}{4\pi f_{max}^2} \tag{3.15}$$

3.2.6 Generating Surface Normals

Using just a height field to render the ocean surface makes the result lacklustre. Therefore, surface normals need to be generated. Instead of using a finite difference approach to approximate surface normals, they can be derived directly by computing the surface's gradient [Tes01] with two additional FFT in a process called spectral differentiation:

$$\nabla h(\boldsymbol{x}, t) = \sum_{\boldsymbol{k}} i\tilde{h}(\boldsymbol{k}, t) \exp(i\boldsymbol{k} \cdot \boldsymbol{x}) \tag{3.16}$$

A tangent to the surface is $(1{,}0{,}\nabla h_x)^T$, another is $(0{,}1{,}\nabla h_y)^T$ thus, the surface's normal can be obtained by computing the cross product and normalizing the result [LeB+12]. Here, the z-axis is the up-vector. When working with multiple bands, their tangents can be added to obtain the combined surface normal [LeB+12].

Regarding quality and efficiency, Tessendorf states that the finite difference approximation requires less memory but is inaccurate for waves with small wavelengths, while the slope vector computed with the FFT is an exact computation at the cost of additional FFT's [Tes01].

3.2.7 Horizontal Surface Displacement

Without any further modifications, the height field produced by the FFT approach shows very smooth and round crests just like the swell in figure 3.7. For a swell, this is adequate, but waves of a local wind sea are bumpy with sharp crests, especially in bad weather conditions [Bee97, p. 59; Tes01].

Tessendorf [Tes01] describes a way to create such choppy waves by creating a vector $\boldsymbol{D} = (D_x, D_y)$ that displaces the surface point horizontally $\boldsymbol{x} + \lambda_d \boldsymbol{D}(\boldsymbol{x}, t)$ using two additional FFTs:

$$\boldsymbol{D}(\boldsymbol{x}, t) = \sum_{\boldsymbol{k}} -i \frac{\boldsymbol{k}}{k} \tilde{h}(\boldsymbol{k}, t) \exp(i \boldsymbol{k} \cdot \boldsymbol{x}) \tag{3.17}$$

The parameter λ_d allows controlling the scaling of the displacement and needs to be adjusted carefully since displacing the positions strongly can result in the surface points intersecting themselves at the top of crests. Besides sharpening crests and stretching troughs, this describes a circular motion of waves which improves the realistic impression of the ocean's animation.

The displacing transformation of the surface point can be further used to detect possible locations of breaking waves, spray or foam. Tessendorf recommends calculating the following Jacobian matrix $\boldsymbol{J}(\boldsymbol{x}, t)$ and its determinant $J = \det(\boldsymbol{J})$ for said detection:

$$\boldsymbol{J}(\boldsymbol{x}, t) = \begin{bmatrix} J_{xx} & J_{xy} \\ J_{yx} & J_{yy} \end{bmatrix} = \begin{bmatrix} 1 + \lambda_d \dfrac{\partial D_x(\boldsymbol{x}, t)}{\partial x} & \lambda_d \dfrac{\partial D_x(\boldsymbol{x}, t)}{\partial y} \\ \lambda_d \dfrac{\partial D_y(\boldsymbol{x}, t)}{\partial x} & 1 + \lambda_d \dfrac{\partial D_y(\boldsymbol{x}, t)}{\partial y} \end{bmatrix} \tag{3.18}$$

Figure 3.8 shows the wave profile upon displacing surface points and its effect on the Jacobian determinant. Using the Jacobian determinant with some threshold function allows detecting sharp crests.

Correcting the surface normal

It is easy to imagine that displacing the surface to create sharp waves should have an effect on the surface normal. Surprisingly, the only mention of such a correction was found in the master thesis by Gamper [Gam18, p. 65] who in turn mentions as a source the private communication with Jonathan Dupuy and Eric Bruneton, from the latter comes the previously cited article [BNH10].

Thus, the surface's slope can be corrected using the J_{xx} and J_{yy} components of the Jacobian matrix $\boldsymbol{J}$. Transferred to the introduced notation

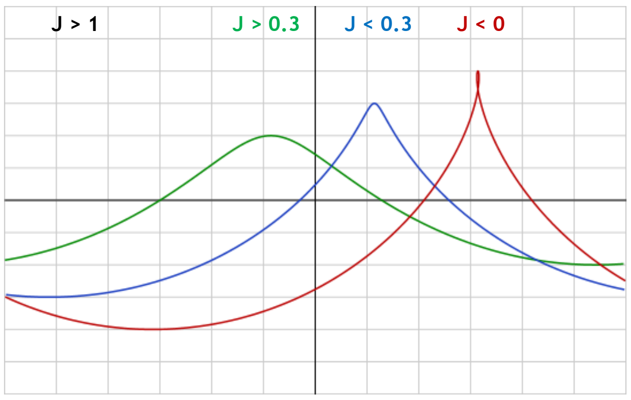


Figure 3.8: Illustration by Tcheblokov [Tch15] showing the effect of displacing the surface point on the wave profile and associated values of the Jacobian determinant J.

of tangents $(1,0,\nabla h_x)^T$ and $(0,1,\nabla h_y)^T$ in section 3.2.6, they can be corrected as follows:

$$\boldsymbol{t}_x = \begin{pmatrix} 1 \\ 0 \\ (\nabla h_x)/(J_{xx}) \end{pmatrix} \quad \boldsymbol{t}_y = \begin{pmatrix} 0 \\ 1 \\ (\nabla h_y)/(J_{yy}) \end{pmatrix} \tag{3.19}$$

3.3 Simulating Shallow Water

To introduce the topic of shallow water simulation, the connection between the Navier-Stokes and shallow water equations is presented, following the book *Fluid Simulation for Computer Graphics* by Bridson [Bri08]. The Navier-Stokes equations describe the motion of fluids. For an incompressible, fluid they include the momentum equation 3.21 and the continuity equation 3.20 which expresses the incompressibility condition:

$$\nabla \cdot \boldsymbol{u} = 0 \tag{3.20}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \frac{1}{\rho}\nabla p = \boldsymbol{g} + \nu \nabla^2 \boldsymbol{u} \tag{3.21}$$

In these equations $\boldsymbol{u}$ is the velocity, ρ is the density of the fluid, p is the pressure which the fluid exerts on its surroundings, and ν is the kinematic viscosity which intuitively measures how much the fluid resists deformation.

From a Lagrangian point of view, one can interpret the momentum equation as different forces acting on a particle of the fluid. There is an external force due to gravity, a force due to pressure differences and a force due to viscosity that models resisting deformations. A particle is able to carry quantities through the fluid with the temperature as a good example. This movement of a quantity through the fluid's velocity field is called advection. Even if the temperature of a single particle flowing through the fluid does not change, measuring the temperature at a fixed point in space may show a change of temperature. The term $(\partial \boldsymbol{u})/\partial t$ expresses the rate of change of the quantity at a fixed point in space while the term $\boldsymbol{u} \cdot \nabla \boldsymbol{u}$ describes how much of that change is due to the fluid's flow through that point. Here, the quantity is not the temperature but the velocity itself.

Considering the strict requirements for a real-time gaming application, numerically solving above equations in 3D for a large simulation domain is still out of scope for current generation of hardware. Hence, lowering the simulation dimension from 3D to 2D is crucial as it naively results in a complexity reduction from $O(n^3)$ to $O(n^2)$.

The driving idea for the shallow water equations is that water of relatively small depth is primarily characterized by horizontal motion [Zho04, p. 10] as there is simply not enough space for vertical motion to have a great effect on the surface. Thus, any differences of forces along the fluid's depth can be neglected and vertical accelerations in the velocity field can be ignored. Instead, the velocity is considered to be *depth-averaged* and expressed just in terms of a horizontal velocity $\boldsymbol{u} \in \mathbb{R}^2$ alongside the direct simulation of the water's depth d.

For the vertical direction of the momentum equation 3.21, this assumption means that the pressure gradient and gravity are much higher than the other terms which, therefore, can be dropped [Bri08, p. 170]. Applying the boundary condition that the pressure p at the interface between the water and the airequals the atmospheric pressure p_a, which can be set to 0, allows for the pressure to be calculated directly [Zho04, p. 15]. This is called the hydrostatic pressure approximation which saves the step of solving for pressure [Bri08, p. 171].

Since the horizontal velocity $\boldsymbol{u}$ is depth-averaged, the advection of the velocity in the momentum equation 3.21 is reduced to just two dimensions. In addition there is an acceleration due to external forces like the gravity [Bri08, p. 171] and the viscosity, if it is not dropped to model ideal inviscid fluids. Instead of the vertical velocity, the depth of the water d will be simulated, i.e. the depths' rate of change and its advection by the horizontal velocity [Bri08, p. 173].

The resulting shallow water equations again include the momentum equation and continuity equation which may be written using Einstein's summation convention as in [Zho04]:

$$\frac{\partial d}{\partial t} + \frac{\partial (d u_j)}{\partial x_j} = 0 \tag{3.22}$$

$$\frac{\partial (d u_i)}{\partial t} + \frac{\partial (d u_i u_j)}{\partial x_j} = -g \frac{\partial}{\partial x_i}\left(\frac{d^2}{2}\right) + \nu \frac{\partial^2 (d u_i)}{\partial {x_j}^2} + F_i \tag{3.23}$$

Here, F_i combines several external forces including the bed shear stress τ_b with a friction coefficient C_b or the slope of the seabed whose elevation is z_b:

$$F_i = -gd\frac{\partial z_b}{\partial x_i} - \frac{\tau_{b\,i}}{\rho} \tag{3.24}$$

$$\tau_{b\,i} = \rho C_b u_i \sqrt{u_j u_j} \tag{3.25}$$

To highlight the similarities between the shallow water and the Navier-Stokes equations, they are written in vector notation:

$$\frac{\partial d}{\partial t} + \boldsymbol{\nabla} \cdot d\boldsymbol{u} = 0 \tag{3.26}$$

$$\frac{\partial d\boldsymbol{u}}{\partial t} = -(d\boldsymbol{u} \cdot \boldsymbol{\nabla})\boldsymbol{u} - g\boldsymbol{\nabla}\frac{d^2}{2} + \nu\boldsymbol{\nabla}^2 d\boldsymbol{u} + \boldsymbol{F} \tag{3.27}$$

$$\boldsymbol{F} = -gd\boldsymbol{\nabla} z_b - \frac{\boldsymbol{\tau}_b}{\rho} \tag{3.28}$$

$$\boldsymbol{\tau}_b = C_b \boldsymbol{u}\, \|\boldsymbol{u}\| \tag{3.29}$$

A comprehensible derivation of the shallow water equations can be found in [Bri08] chapter 12 and a full mathematical derivation in [Zho04] chapter 2. Different numerical methods have been proposed to solve previous equations, but this work focuses on the LBM.

3.3.1 Lattice Boltzmann Method for Shallow Water Equations

In general, the Lattice Boltzmann Method allows simulating different macroscopic physical phenomena with a microscopic (to be precise, mesoscopic [Moh19, p. 3]) model of particles moving and colliding with each other on restricted lattice directions [Zho04, p. 33]. These particles can be considered virtual particles, since their movement and interactions are of statistical nature described by probability distribution functions. While the LBM has been used to solve a wide array of problems in one, two or three dimensions, this work focuses on the 2D shallow water equations. Please refer to *Lattice Boltzmann Method* [Moh19] for an overview of possible applications.

Defining a lattice pattern

There exist different lattice patterns for the LBM which usually follow the naming convention of DxQy with x for the dimension and y for the quantity of lattice connections. Figure 3.9 shows the square D2Q9 pattern used in this work, which Zhou suggests to use for increased accuracy and ease of use [Zho04, p. 21]. The lattice defines the available movement directions of particles and reflects to the spatial domain with a lattice size Δx. For each of the movement directions $i \in [0, 8]$, there exists a particle velocity vector $\boldsymbol{e}_i$

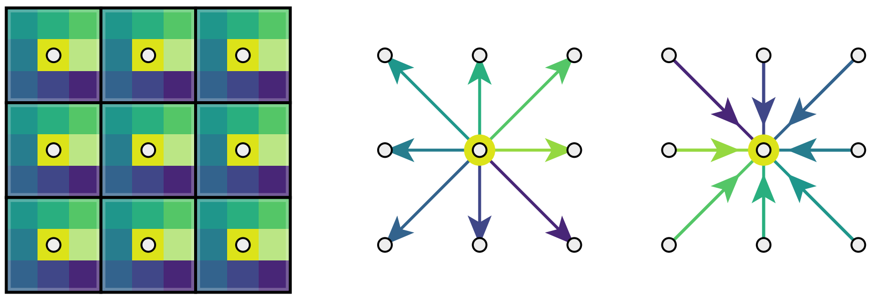


Figure 3.9: The streaming (middle) and collision operation (right) of the Lattice Boltzmann Method are illustrated with the D2Q9 lattice. The grid structure on the left shows that for each lattice direction the value of the probability distribution function needs to be stored.

with e as the particle speed:

$$\{\boldsymbol{e}_i\} = \left\{ \begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}e\\0\end{pmatrix}, \begin{pmatrix}e\\e\end{pmatrix}, \begin{pmatrix}0\\e\end{pmatrix}, \begin{pmatrix}-e\\e\end{pmatrix}, \begin{pmatrix}-e\\0\end{pmatrix}, \begin{pmatrix}-e\\-e\end{pmatrix}, \begin{pmatrix}0\\-e\end{pmatrix}, \begin{pmatrix}e\\-e\end{pmatrix} \right\} \tag{3.30}$$

Stream and collision step

The basic algorithm consists of two steps: the stream step and the collision step [Zho04, p. 19]. During the stream step (see figure 3.9 middle) each lattice point moves its current particle distribution functions $f_i{}'$ to the according neighbouring lattice point, which in its most basic form excluding external forces can be written as follows [Oje13a, p. 19]:

$$f_i\left(\boldsymbol{x} + \boldsymbol{e}_i \Delta t, t + \Delta t\right) = f_i{}'\left(\boldsymbol{x}, t\right) \tag{3.31}$$

The second step is the collision step (see figure 3.9 right), where the arriving particle distribution functions at a lattice point collide as expressed by the collision operator Ω:

$$f_i{}'\left(\boldsymbol{x}, t\right) = f_i\left(\boldsymbol{x}, t\right) + \Omega_i\left(f_i\left(\boldsymbol{x}, t\right)\right) \tag{3.32}$$

This collision operator is a very general description and different operators exist, some of which may become very complicated and take the form of a complex matrix [Zho19a]. The BGK collision operator (named after their inventors Bhatnagar, Gross and Krook) is simple and efficient and, therefore, widely used [Zho04; Moh19]:

$$\Omega_i = -\frac{1}{\tau}\left(f_i - f_i^{eq}\right) \tag{3.33}$$

Here, τ is the single relaxation time which is related to the fluids viscosity (see section 3.3.1) and f_i^{eq} is the local equilibrium distribution function which will be explained shortly. Both streaming and collision step are usually written as a single equation referred to as the Lattice Boltzmann equation:

$$f_i\left(\boldsymbol{x} + \boldsymbol{e}_i \Delta t, t + \Delta t\right) = f_i\left(\boldsymbol{x}, t\right) - \frac{1}{\tau}\left(f_i - f_i^{eq}\right) \tag{3.34}$$

The local equilibrium distribution function

To actually model the shallow water equations with above equation, the local equilibrium distribution function needs to be defined properly. It is the core of the method and other definitions can be used to model other equations, for example the Navier-Stokes equations. For the SWE, the widespread definition obtained from [ZL13] and rewritten in vector notation instead of Einstein summation notation is:

$$f_i^{eq} = \begin{cases} d\left(1 - \dfrac{5gd}{6e^2} - \dfrac{2\boldsymbol{u}\cdot\boldsymbol{u}}{3e^2}\right) & , i = 0 \\ \lambda_i d\left(\dfrac{gd}{6e^2} + \dfrac{\boldsymbol{e}_i\cdot\boldsymbol{u}}{3e^2} + \dfrac{(\boldsymbol{e}_i\cdot\boldsymbol{u})^2}{2e^4} - \dfrac{\boldsymbol{u}\cdot\boldsymbol{u}}{6e^2}\right) & , i \neq 0 \end{cases} \tag{3.35}$$

$$\text{with } \lambda_i = \begin{cases} 1 & i = 1,3,5,7 \\ 1/4 & i = 2,4,6,8 \end{cases} \tag{3.36}$$

Adding external forces

So far external forces were omitted for the sake of clarity. Throughout several works [Zho11; ZL13], Zhou refined the evaluation of forces and this work adopts the latest notation from [Zho19b]:

$$\begin{aligned} f_i\left(\boldsymbol{x} + \boldsymbol{e}_i \Delta t, t + \Delta t\right) = &f_i\left(\boldsymbol{x}, t\right) - \frac{1}{\tau}\left(f_i - f_i^{eq}\right) \\ &- \frac{g\overline{d}}{e^2} C_i\left(z_b\left(\boldsymbol{x} + \boldsymbol{e}_i \Delta t\right) - z_b\left(\boldsymbol{x}\right)\right) + C_i \frac{\Delta t}{e^2} \boldsymbol{e}_i \cdot \boldsymbol{F} \end{aligned} \tag{3.37}$$

$$\text{with } \overline{d} = \frac{1}{2}\left(d(\boldsymbol{x} + \boldsymbol{e}_i \Delta t, t) + d(\boldsymbol{x}, t)\right) \tag{3.38}$$

$$\text{and } C_i = \begin{cases} 0 & i = 0, \\ 1/3 & i = 1,3,5,7, \\ 1/12 & i = 2,4,6,8 \end{cases} \tag{3.39}$$

In this definition, the seabed elevation is accurately embedded into the equation with a semi-implicit form for $\overline{d}$ that saves deriving the seabed for its slope [Zho11]. Thus, force term $\boldsymbol{F}$ just includes bed sheer stress, wind sheer stress and Coriolis force due to earth's rotation [ZL13]. However, only bed sheer stress $\boldsymbol{\tau}_b$ (see equation 3.29) is considered in this work since Coriolis force has no great visual significance, and the wind interaction is modelled by deep water which will be coupled to the LBM.

Calculating depth and horizontal velocity

Applying the steps mentioned up to this point evolves 9 distribution functions of said virtual particles which all have to be stored per lattice point (see figure 3.9 left). The final step to make practical use of them is to calculate the macroscopic quantities of depth d and horizontal velocity $\boldsymbol{u}$ of the Shallow Water equations:

$$d(\boldsymbol{x}, t) = \sum f_i(\boldsymbol{x}, t) \tag{3.40}$$

$$d(\boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{x}, t) = \sum \boldsymbol{e}_i f_i(\boldsymbol{x}, t) \tag{3.41}$$

$$\Rightarrow \boldsymbol{u} = \frac{d(\boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{x}, t)}{d(\boldsymbol{x}, t)} \tag{3.42}$$

Towards stable simulation

The Chapman-Enskog expansion can be used on the Lattice Boltzmann equation 3.37 to prove that the calculated water depth and horizontal velocity are a solution to the shallow water equations 3.26 and 3.27. This proof is not repeated in this work as it is already presented in [Zho04] and [Zho11] in detail. The proof unveils that the kinematic viscosity ν of the fluid is defined by:

$$\nu = \frac{e^2 \Delta t}{6} (2\tau - 1) \tag{3.43}$$

According to Zhou, in general the method is numerically stable when it satisfies four conditions [Zho04]:

$$\tau > \frac{1}{2} \tag{3.44}$$

$$\frac{\boldsymbol{u} \cdot \boldsymbol{u}}{e^2} < 1 \tag{3.45}$$

$$\frac{gd}{e^2} < 1 \tag{3.46}$$

$$Fr = \frac{\sqrt{\boldsymbol{u} \cdot \boldsymbol{u}}}{\sqrt{gd}} = \frac{\boldsymbol{u} \cdot \boldsymbol{u}}{gd} < 1 \tag{3.47}$$

The relaxation time condition ensures the kinematic viscosity to be positive. Then, neither the magnitude of velocity nor the celerity $\sqrt{gd}$ (see the shallow water approximation for the dispersion relation in equation 3.1) can exceed the particle speed. And lastly, the flow needs to be subcritical, as expressed by the *Froude number Fr* < 1.

Still, for very small viscosities, usually due to the single time relaxation having values close to its limit, the method can become numerically unstable [Zho19a]. Several improvements have been proposed to remedy this issue. Zhou suggests using a Smagorinsky sub-grid model which enables simulating the flow at an eddy viscosity independent of the fluids characteristic kinematic viscosity, which is adopted by [Oje13a]. Other approaches like Two-Relaxation-Time Scheme or Multi-Relaxation-Time scheme modify the collision operator [Moh19, p. 145−149] but come with an increased computational cost [Zho19a].

The key observation regarding stability is its connection to the collision operator 3.33. Recently, Zhou proposed a simplification of the Lattice Boltzmann equation that allows to remove the collision operation by setting the single relaxation time τ to 1. As a result, the stability requirements are clearly defined and the macroscopic properties of depth and velocity can be used directly. Both are promising advantages for the usage in context of computer graphics.

3.3.2 Macroscopic Lattice Boltzmann method for SWE

To get to a macroscopic formulation of the LBM, the Lattice Boltzmann equation needs to be rewritten following [Zho19b]. For the sake of clarity, equation 3.34 will be used as basis in this work rather than equation 3.37 since the force related terms can be taken over. Rewriting equation 3.37 yields:

$$\begin{aligned} f_i(\boldsymbol{x}, t) = &f_i(\boldsymbol{x} - \boldsymbol{e}_i \Delta t, t - \Delta t) \\ &- \frac{1}{\tau}\Big(f_i(\boldsymbol{x} - \boldsymbol{e}_i \Delta t, t - \Delta t) - f_i^{eq}(\boldsymbol{x} - \boldsymbol{e}_i \Delta t, t - \Delta t)\Big) \end{aligned} \tag{3.48}$$

The purpose of the collision is the relaxation of the local particle distribution function towards its local equilibrium [Zho19a]. By setting the single relaxation time τ to 1, this relaxation reduces to the local equilibrium distribution function itself:

$$\begin{aligned} f_i(\boldsymbol{x}, t) = &f_i(\boldsymbol{x} - \boldsymbol{e}_i \Delta t, t - \Delta t) \\ &- \Big(f_i(\boldsymbol{x} - \boldsymbol{e}_i \Delta t, t - \Delta t) - f_i^{eq}(\boldsymbol{x} - \boldsymbol{e}_i \Delta t, t - \Delta t)\Big) \\ f_i(\boldsymbol{x}, t) = &f_i^{eq}(\boldsymbol{x} - \boldsymbol{e}_i \Delta t, t - \Delta t) \end{aligned} \tag{3.49}$$

Equation 3.49 is solely based on the physical variables of velocity and depth instead of requiring particle distribution functions of previous time steps.

Reintegrating the force terms yields:

$$\begin{aligned} f_i(\boldsymbol{x},t) =& f_i^{eq}(\boldsymbol{x}-\boldsymbol{e}_i\Delta t, t-\Delta t) \\ &- C_i\frac{g\bar{d}}{e^2}\left(z_b(\boldsymbol{x}) - z_b(\boldsymbol{x}-\boldsymbol{e}_i\Delta t)\right) \\ &+ C_i\frac{\Delta t}{e^2}\boldsymbol{e}_i\cdot\boldsymbol{F} \end{aligned} \tag{3.50}$$

Parametrization under the absence of collision

The absence of the collision operation reduces the eddy viscosity ν to just:

$$\nu = \frac{e^2\Delta t}{6} \tag{3.51}$$

To regain some level of control over the viscosity, Zhou suggests determining the particle speed e in terms of ν and Δx with the time step Δt depending on Δx and e, such that:

$$\Delta t = \frac{\Delta x}{e} \qquad \Rightarrow \nu = \frac{e\Delta x}{6} \qquad \Rightarrow e = \frac{6\nu}{\Delta x} \tag{3.52}$$

Thus, for a given viscosity ν the only parameter required to control the simulation is the lattice size Δx.

Stability conditions and evaluation

Previous established stability conditions (see 3.3.1) remain intact, naturally with the omission of the one regarding the single relaxation time. Additionally, Zhou finds that the Lattice Reynolds number must be less than 1 [Zho19a]. The Reynolds number can be used to predict if a flow is laminar or turbulent [Bee97, p. 181] and is defined as [Moh19, p. 108]:

$$Re = \frac{U\,L}{\nu} \tag{3.53}$$

Here, U and L are the flow speed and spatial length in macroscopic scale [Moh19, p. 108]. The Lattice Reynolds number on the other side is defined for the lattice and for the macroscopic LBM can be written as follows [Zho19a]:

$$Re_{\mathrm{L}} = \frac{U\,\Delta x}{\nu} \tag{3.54}$$

The only option to match a flow Reynolds number on the lattice with the simulation staying stable is, to increase the grid size n because the relation is $n = L/(\Delta x)$. Taking U as the maximum flow speed, the limit $Re_{\mathrm{L}} < 1$ can be written as a limit on velocity $\boldsymbol{u}$:

$$\sqrt{\boldsymbol{u} \cdot \boldsymbol{u}} \frac{\Delta x}{\nu} < 1 \tag{3.55}$$

Here, the fundamental limitation arises that for high velocities or a small eddy viscosity, the lattice size Δx becomes very small upon satisfying the stability conditions. Thus, to simulate a large spatial domain or a high Reynolds number, the required grid size n quickly becomes unsustainable. Further discussion of problems due to stability conditions and the parametrization along with a proposal on how to deal with them can be found in section 4.2.1.

The prevalent upside of the macroscopic LBM is the direct usage of the macroscopic physical quantities. From a programming perspective, it naively reduces the memory usage from 9 floating-point values to just one for depth and two for the horizontal velocity. It also spares the definition of boundary conditions in terms of particle distribution functions since they can be directly applied to the physical quantities [Zho19b].

3.4 Surface Mesh

Multiple height fields are the result of deep water simulation, whereas the shallow water simulation outputs depths that need to be combined with the seabed elevation to obtain heights. Both results need to be sampled in some way to render the ocean surface. For real-time applications a mesh-based approach is the typical choice. While the primary view is a top-down perspective, occasional close-up views including a visible horizon need to be supported as well.

During development, a simple uniform grid was used as a starting point, but it clearly is unsuitable for a large scale ocean. Hence, some kind of spatially adaptive technique needs to be employed. Spatial data structures or mesh refinement techniques to increase mesh resolution in regions of interest are promising but complicate the implementation. The projected grid concept of Johanson [Joh04] is a rather tempting and widely adopted idea due to the prospect of having a naturally adaptive grid restricted to the camera frustum.

3.4.1 Algorithm of the Projected Grid

The core idea of the projected grid approach is to define a uniform grid in screen space and project it onto the horizontal plane in world space. The algorithm is now presented at a high level with the help of figures 3.10, 3.11 and 3.12. Please refer to [Joh04] for mathematical details.

First and foremost a basis seaplane (see figure 3.10) needs to be defined that matches the sea level at rest. A grid will be projected on this plane with

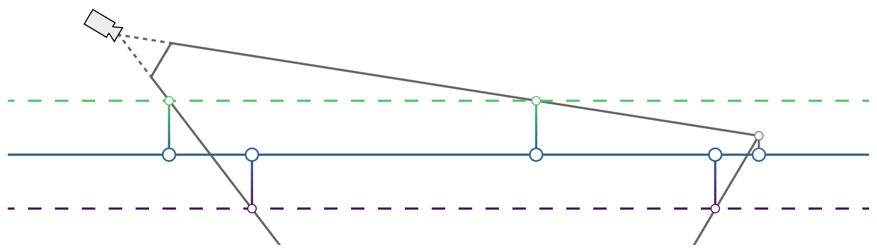


Figure 3.10: The intersections between the frustum's edges and the upper and lower seaplane as well as the frustum corner point inside the spanned volume are projected onto the basis seaplane.

the grid points as the vertices of a mesh. Values of the height field will be used to offset these vertices along the seaplane's up facing normal. Without precautions, this displacement may result in edges of the mesh being visible inside the camera's frustum. Therefore, two more planes above and below the basis seaplane are defined at a distance matching the maximum possible displacement of the sea surface.

Using the frustum's corner points in world space coordinates, the intersections of the frustum's edges with the upper and lower seaplane are calculated. Together with corner points lying inside the volume spanned by both planes, they are projected onto the basis seaplane as depicted in figure 3.10).

Some camera views raise issues, for instance, if the view direction is parallel or pointing away from to the plane or if the projection of the seaplane intersects the near plane. Johanson suggests using a second adjusted camera for the upcoming projection, called the projector [Joh04]. A suitable projector can be found by ensuring its location stays above the upper plane and its direction faces the seaplane. Figure 3.11 shows a possible projector for the used camera.

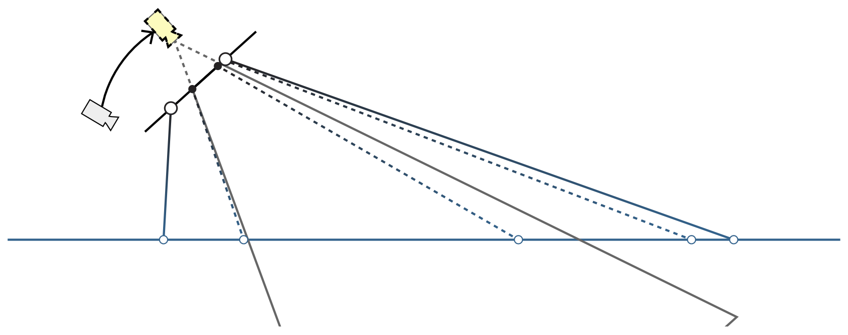


Figure 3.11: The obtained points from figure 3.10 are projected into screen space of a projector camera. In this space the minimum and maximum can be determined.

Once the projector is found and its view and projection matrices calculated, all the world space points previously projected on the seaplane are then projected into the projectors screen space. As depicted in figure 3.11, this step is required to find the screen space minimum and maximum in terms of the points' x and y components. Projecting them back into world space yields the minimum and maximum coordinates of the grid.

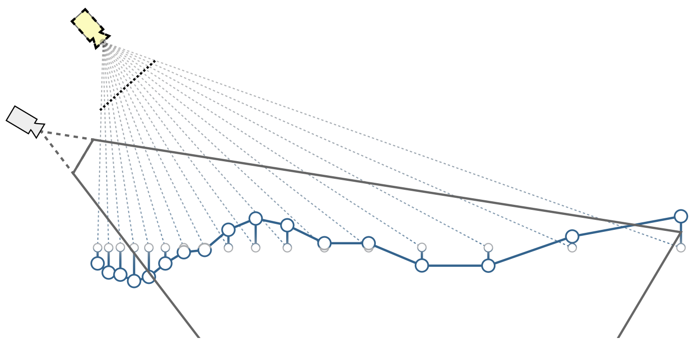


Figure 3.12: By projecting the screen space grid points back to world space the projected grid is obtained. Its world space coordinates are displaced according to the height field.

How to use the found minimum and maximum points is a question of implementation. Johanson uses them to create a conversion matrix for uv-coordinates. In this work, the projected homogeneous world space coordinates for the grid corners are instead used for interpolation. It should be noted that both methods use homogeneous world space coordinates as it is fundamental for the resulting grid to be uniform in screen space. Thus, after obtaining the homogeneous grid points on the projectors view plane the perspective division moves them on the seaplane. Figure 3.12 displays this step followed by the displacement along the seaplane's normal weighted with heights obtained by sampling the height field. It can be seen that the resulting mesh rendered by the original camera lies completely inside its frustum.

3.4.2 Problems of Projected Grid Approach

The major problem of the projected grid approach is the swimming artefact. It can be easily thought of by considering the camera in figure 3.12 moving horizontally. As the world positions of grid points move with the camera, different height values are sampled which were not represented by the mesh previously. This gets worse, the further away samples are taken.

To properly define the upper and lower bound of the seaplane, the maximum height offset has to be known. Additionally, horizontal displacement as obtained by the FFT approach is not considered.

If the camera is inside the spanned volume, the projector quickly ends up being repositioned in a way that the resulting grid approaches a rectangular shape in world space [Joh04]. In this case, the benefit of closely matching the mesh to the frustum is lost.

3.4.3 Possible Improvements

Bruneton, Neyret, and Holzschuch [BNH10] use the projected grid in combination with three different surface detail levels. Displacement to the surface is only applied close to the camera and faded out at a certain threshold. Normals are faded out further away while lighting with a BRDF is faded in simultaneously. This renders the swimming artefact less visible but does not eliminate it.

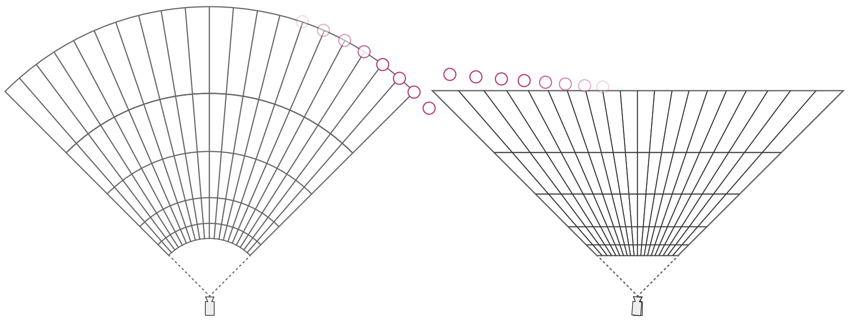


Figure 3.13: Polar meshing improvement (left) proposed in [Bow13] compared to standard projected grid (right).

A more sophisticated approach to fight the vertex swimming is presented at Special Interest Group on Graphics and Interactive Techniques (SIGGRAPH) 2013 by Bowles [Bow13]. It is suggested to arrange the vertices in a circle around the camera and snap the rotation angles. Thus, vertex positions stay fixed during rotation which does not hold true for the standard projected grid (see figure 3.13). To fix vertices during a forward motion, Bowles proposes a split-and-merge scheme inspired by a quad tree compression [Bow13]. However, it stays impossible to fix vertices during sideways camera motions.

Kryachko presents findings on the projected grid approach from the development of the video game *World of Warships* (2015) during SIGGRAPH 2016 [Kry16]. Thus, increasing the vertical mesh resolution and adding randomness to the grid vertices is an improvement but the swimming (here called shaking) artefact remains noticeable. In the end, Kryachko opts to use the projected grid only as an initial approximation which is refined with height field ray casting on a per pixel level [Kry16].

4 Implementation

Development was done in *Epic Games'* Unreal Engine (UE) of version 4.25. UE provides an abstraction layer for multiple graphic application programming interfaces (APIs) called the *Render Hardware Interface* [Epi] which mostly follows the terminology of *Direct3D* with shader code written in High Level Shading Language (HLSL). This work's implementation is presented from a functional point of view without any UE specifics. It shall be mentioned that UE's deferred rendering pipeline was used and graphical commands are queued into a command list which is translated to Direct3D 12 rendering commands one frame later [Epi].

Throughout this work the acronyms central processing unit (CPU) and GPU are used to indicate whether work is performed by the host or the device respectively.

Figure 4.1 shows a flow chart of the implementation split in CPU and GPU work and a highlighting of the five main features: deep water simulation, shallow water simulation with LOD system, projected grid surface mesh, simulation output combination and ocean-object interactions.

Deep water simulation employs a multi-band and multi-spectra FFT approach and is fully implemented on the GPU, including the sampling of wave spectra.

Simulation of shallow water is done with the LBM. A theoretical description of a LOD system is proposed, and details on a possible implementation are presented. Additionally, the plausible coupling of the deep water FFT approach with the shallow water simulation is shown.

The macroscopic simulation of physical quantities allows the interaction of rigid bodies with the shallow water simulation in an efficient and straightforward manner. Reading back the buoyant force from the GPU to the CPU is implemented in a non-stalling way. Obtained forces are forwarded to the engines physics system.

While UE deals with the rendering of the ocean surface, an emphasis is put on combining the great amount of simulation outputs due the shallow water LOD system and the multi-band, multi-spectra FFT approach.

The surface mesh is created using a version of projected grid approach with a proposal to reduce the swimming artefact.

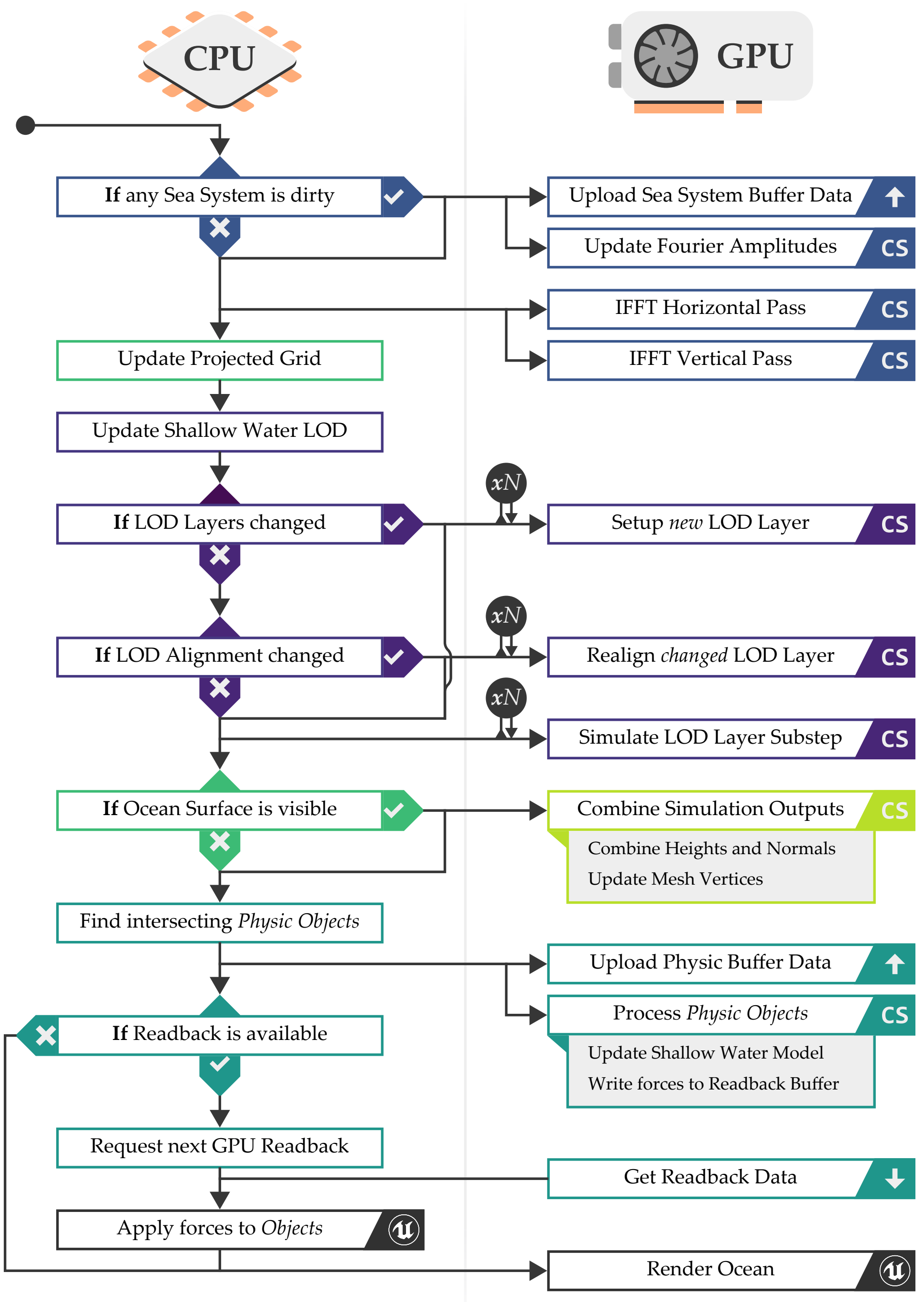

CPU
GPU
If any Sea System is dirty
Upload Sea System Buffer Data
Update Fourier Amplitudes
CS
IFFT Horizontal Pass
IFFT Vertical Pass
Update Projected Grid
Update Shallow Water LOD
xN
If LOD Layers changed
Setup *new* LOD Layer
If LOD Alignment changed
Realign *changed* LOD Layer
Simulate LOD Layer Substep
If Ocean Surface is visible
Combine Simulation Outputs
Combine Heights and Normals
Update Mesh Vertices
Find intersecting *Physic Objects*
Upload Physic Buffer Data
Process *Physic Objects*
Update Shallow Water Model
Write forces to Readback Buffer
If Readback is available
Request next GPU Readback
Get Readback Data
Apply forces to *Objects*
Render Ocean

Figure 4.1: High level overview of the implementation.

4.1 Simulating Deep Water

The presented deep water simulation is based on Park and Park's multi-spectra approach to model different wave systems [PP20] and LeBlanc et al. multi-band approach [LeB+12]. In addition, own insights and recommendations on moving most of the work to GPU and modifying the sea state at runtime are contributed.

4.1.1 Controllability

To render the approach useful and interesting in a game environment, it has to offer the possibility of modifying the sea state at runtime naturally without showing hard transitions. For some parameters, this means special care has to be taken, e.g. changes of wind speed may alter wave heights but not reveal the tiling of height fields. This has implications on spectrum sampling and is shown in section 4.1.2. Other parameters cannot be changed at runtime either because it is impossible to achieve smooth transitions or due to technical reasons. In the context of this work, former are called *dynamic* parameters and latter *static* parameters.

Impact of parameters on performance

Regarding performance and memory requirements, three crucial aspects are important. First, the size of the Fourier grid N that directly translates to the height field texture size. Secondly, the amount of sea systems and the number of frequency bands they consist of. Finally, the types of features which are required for bands of the sea systems. These features include the generation of normals either by spectral differentiation or central differences and horizontal displacements. It is advisable to allow choosing features on a per-band basis because for example low frequency swells have round crests that do not require to be displaced. All these options are static parameters and fixed at start-up. In section 5.1.1 a closer look at their impact on performance can be found.

Wave spectra parameters

As in [PP20], two spectral functions are supported. The JONSWAP spectrum is controlled with the wind speed U and wind fetch F. Depicting certain weather conditions may be facilitated by using the *Beaufort Scale* as reference. With the peak period T_p and the significant wave height H_s, the JONSWAP-Glenn spectrum describes the sea state in a more direct way and is suited to control the low frequency swell in particular. Both spectra are combined with directional spreading on a per sea system basis, controlling the wave's propagation direction with the help of a wind direction $\boldsymbol{u}_w$.

4.1.2 Sampling Wave Spectra

As introduced in section 3.2.2, the FFT requires time-dependent amplitudes $\tilde{h}(\boldsymbol{k}, t)$ which in turn require time-independent amplitudes $\tilde{h}_0(\boldsymbol{k})$. Latter $\tilde{h}_0(\boldsymbol{k})$ do not change with time and can be calculated, stored and used until parameters of the directional wave spectrum S, the gravity g or the wave vectors $\boldsymbol{k}$ change. Wave vectors $\boldsymbol{k}$ define the sampling locations of the spectral densities with the dispersion relation $\sqrt{gk}$, therefore, their definition is fundamental. In order to choose the wave vectors, the spectra itself need to be taken into account, as their graphs vary with the change of parameters.

The problem of run-time sampling

Both JONSWAP and JONSWAP-Glenn spectra use a peak frequency ω_p or f_p expressing where the graph's peak is located. It depends on the spectrum parameters such that:

$$\omega_p = 22 \left(\frac{g^2}{U_{10} F} \right)^{1/3} \tag{4.1}$$

$$f_p = \frac{1}{T_p} \tag{4.2}$$

As most energy exists around this peak, Park and Park suggest choosing the main sampling range in its vicinity with an upper bound at $2.5 f_p$ [PP20]. LeBlanc et al. choose a fixed upper bound for the highest frequency band matching the wave length of 12 cm and a multiple of the peak frequency for the lowest frequency band. Both share the same problem that arises when the peak frequency is changed at runtime. Changing the peak frequency stretches or compresses the graph horizontally. Describing the sampling range in dependence on the peak frequency results in the unnatural animation of the ocean's surface height field obtained with the FFT. Reasons for this are the non-fixed sampling locations and the dependency of the height field's spatial length on the sampling range. Park and Park recognize the problem of changing the sea state at runtime and suggest keeping the period and phase stationary in Fourier domain [PP20]. However, no further details on how to achieve this are given.

Band division and fixed spectrum sampling

This work's proposed solution to the sampling problem is to fix the sampling range and locations at start-up. Thus, two sets of parameters are defined for each spectrum. The dynamic wind speed U, dynamic wind fetch F, dynamic peak period T_p and dynamic significant wave height H_s can be modified at runtime. The static wind speed U^{st}, static wind fetch F^{st}, static

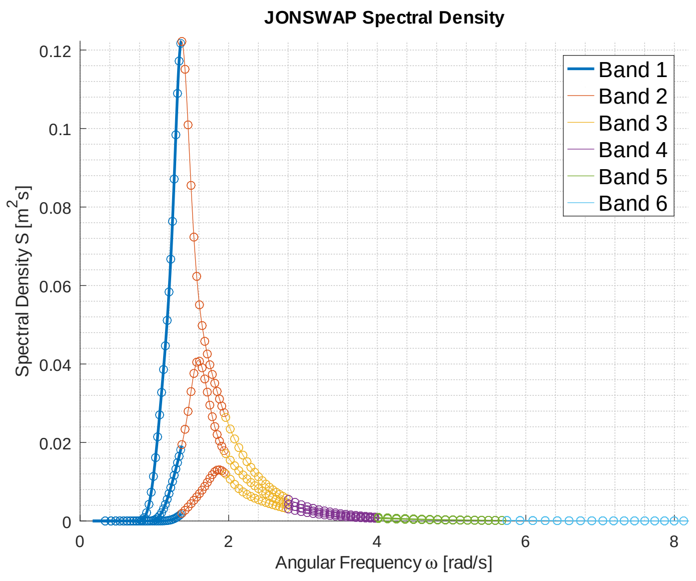


Figure 4.2: Fixed spectrum sampling where the graph with the highest peak matches the static peak frequency. With $\gamma_{\mathrm{s}} = 0$ Band 1 exactly aligns with the spectrum's peak because $p_{\mathrm{L}} = 1$. Band 2 to Band 6 sample the spectrum up to f_{U} with an equal distribution of samples. The graphs with lower peaks differ in their dynamic peak, however, due to the fixed sampling strategy the sampling locations are identical. Note that overlapping samples were removed.

peak period T_p^{st} and static significant wave height ${H_s}^{\mathrm{st}}$ are fixed at start-up, and used to determine the sampling range. Additionally, three parameters to control the distribution of the frequency band's sampling intervals are introduced: The peak sampling enhancement factor γ_{s}, the upper bound peak frequency factor p_{U} and the lower bound peak frequency factor p_{L}. The static peak angular frequency ω_p^{st} and static peak frequency f_p^{st} are:

$$\omega_p^{\mathrm{st}} = 22\left(\frac{g^2}{{U^{\mathrm{st}}}_{10}F^{\mathrm{st}}}\right)^{1/3} \tag{4.3}$$

$$f_p^{\mathrm{st}} = \frac{1}{T_p^{\mathrm{st}}} \tag{4.4}$$

Here, frequencies f and angular frequencies ω are only used to distinguish JONSWAP from JONSWAP-Glenn. The relation $\omega = 2\pi f$ can be used for conversion.

Next, a lower bound fitting factor f_{fit} and upper and lower bound frequency $f_\mathrm{U}, f_\mathrm{L}$ are introduced.

$$f_\mathrm{U} = p_\mathrm{U} * f_p^\mathrm{st} \tag{4.5}$$
$$f_\mathrm{L} = p_\mathrm{L} * f_p^\mathrm{st} \tag{4.6}$$
$$f_\mathrm{fit} = \frac{1}{N}\left(\frac{p_\mathrm{L}}{p_\mathrm{U}}\right)^{(-2)/(M-1)} \tag{4.7}$$

Using these, the goal is to find the minimum and maximum frequency for each band $m \in [1, M]$ with $M > 1$ where band $m = 1$ covers the lower frequencies and band $m = M$ the highest. Starting with $f_M^\mathrm{max} = f_\mathrm{U}$, they are defined recursively by:

$$f_{m-1}^\mathrm{max} = f_m^\mathrm{min} \tag{4.8}$$
$$f_m^\mathrm{min} = f_m^\mathrm{max}(N f_\mathrm{fit}\xi)^{-1/2} \tag{4.9}$$
$$\xi = 1.0 + \gamma_\mathrm{s} * \max\left(0, \frac{f_\mathrm{L} - f_m^\mathrm{max}(N f_\mathrm{fit})^{-1/2}}{f_\mathrm{L} - f_\mathrm{U}}\right) \tag{4.10}$$

In the simplest case of $\gamma_\mathrm{s} = 0$, this results in $f_1^\mathrm{max} = f_\mathrm{L}$ with bands $1 < m \le M$ covering the interval $[f_\mathrm{L}, f_\mathrm{U}]$ each with approximately the same of amount of unique sample locations (see figure 4.2).

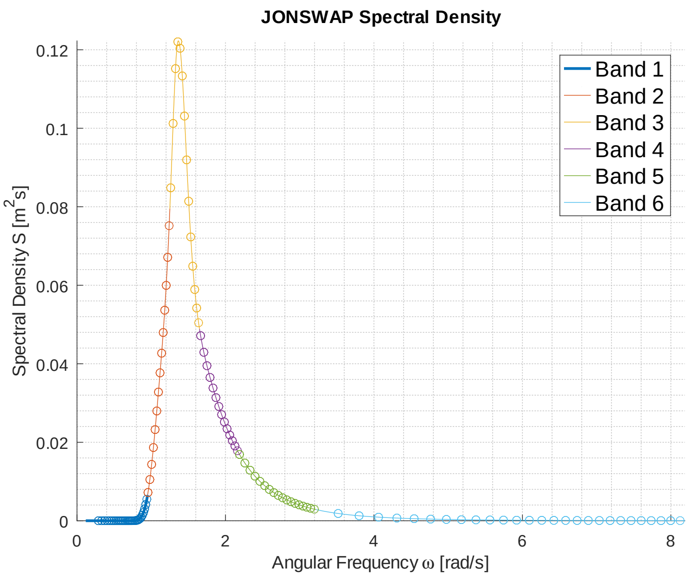


Figure 4.3: The graph shows sampling bounds shifted towards the left with $\gamma_\mathrm{s} = 4$. At the same time the lower bound peak frequency factor was increased $p_\mathrm{L} = 1.5$ to counteract the left shift leading to an increased sample density at the peak.

Increasing γ_s shifts the band's sampling bounds towards the origin and is meant to increase the sampling density around the peak (see figure 4.3). Unfortunately, the property of $f_1^{\max} = f_L$ is lost in the process rendering the placement of samples less intuitive. The increased sampling density also comes at the cost of a reduced number of unique non-overlapping samples.

In both spectrum plots overlapping samples were removed if $f < f_m^{\min}$. Their contribution can be removed during creation of Fourier amplitudes $\tilde{h}_0(\boldsymbol{k})$ by setting their amplitudes to 0 [LeB+12]. In practice, this means, a lot of time is spent on samples that do not have any contribution at all. This is a fundamental difficulty of the multi-band approach because the FFT requires equally distributed samples. Not removing lower band contributions in overlapping sampling areas means over representing low frequencies. For this work, lower band contributions were not removed as the results were satisfying. However, a more in-depth comparison and evaluation of this difficulty should be conducted in future works.

Calculating wave vectors

With a spectrum sampling strategy in place the matching wave vectors need to be calculated on the $N \times N$ grid. As shown before in equation 3.8a, Tessendorf's definition for $\boldsymbol{k} = \left(k_x, k_y\right)$ is:

$$k_x = \frac{2\pi j_x}{L} \text{ with } \frac{-N}{2} \leq j_x < \frac{N}{2} \tag{4.11}$$

$$k_y = \frac{2\pi j_y}{L} \text{ with } \frac{-N}{2} \leq j_y < \frac{N}{2} \tag{4.12}$$

Following Park and Park, the spatial length L only depends on a bands maximum frequency $f^{\max}$ [PP20]. In preparation of the GPU implementation, a restatement preferring a multiplication is suggested. Additionally, the wave vector's components are rearranged as required by the FFT:

$$\omega_m^{\max} = 2\pi f^{\max} \qquad k_m^{\max} = \frac{1}{g}\omega_m^{\max 2} \tag{4.13}$$

$$L_m = \frac{gN}{4\pi f_m^{\max 2}} \qquad \Delta l_m = \frac{L_m}{N} \tag{4.14}$$

$$\lambda_{km} = \sqrt{2} * \frac{k_m^{\max}}{N+1} \qquad \delta(j) = \begin{cases} N, & \text{if} \quad j \geq N/2 \\ 0, & \text{if} \quad j < N/2 \end{cases} \tag{4.15}$$

$$k_m x = \lambda_{km}(j_x - \delta(j_x)) \quad \text{with } 0 <= j_x < N \tag{4.16}$$

$$k_m y = \lambda_{km}(j_y - \delta(j_y)) \quad \text{with } 0 <= j_y < N \tag{4.17}$$

4.1.3 GPU implementation

To reduce the amount of data passed from CPU to GPU to a minimum, both sampling the wave spectra and the FFT is done on the GPU. Figure 4.4 gives an overview of the three involved compute shaders and the texture resources. Updating time-independent Fourier amplitudes is only done when necessary. The Fast Fourier Transform is split in a horizontal and vertical pass and done every frame.

Updating time-independent Fourier amplitudes

During the Fourier amplitude update, three buffers are required: One to hold relevant parameters of sea systems, another one for parameters that are unique per band and finally, an index buffer to map the dispatch's z index to preceding buffers. Amplitudes and wave vectors are written into a $N \times N$ `Texture2DArray` with M slices, one for each band. Here, M is the number of total bands of possibly multiple sea systems. JONSWAP and JONSWAP-Glenn are quite similar, hence, the identification of a band's spectrum type is only required once when sampling the spectrum. For this purpose, the sea system parameter buffer holds the spectrum's type. No further difficulties arise on implementing the spectral functions $\mathrm{S}(\omega)$ on the GPU, but it is naturally recommended to pre-calculate whatever possible.

The directional spreading function $\mathrm{Dir}(\omega, \theta)$ (3.5a) needs more attention because it contains the gamma function $\Gamma = (n-1)!$. Here, the Lanczos approximation [Lan64] is used since its implementation is straight forward and it is very efficient to compute, depending on the choice of coefficients. Using just 4 coefficients obtained from [Totnd] with $g = 3.65$, the simplified HLSL implementation in listing 4.1 was derived, based on code from [Mun20]. It is reasonable to compute on the GPU, and a visual comparison of the proposed approximation with *Octave*'s gamma function shows no differences which is a sufficient mark of quality for the context of this work.

```hlsl
float LanczosGammaApproximation(float s)
{
  const float coefficients[4] =
  { 2.50662846, 41.41740453, -27.06389249, 2.23931796 };
  float lanczosSum = coefficients[3] / (s + 3.0);
  lanczosSum += coefficients[2] / (s + 2.0);
  lanczosSum += coefficients[1] / (s + 1.0);
  lanczosSum += coefficients[0];
  float z = s - 1.0;
  float gamma =  0.0395155 * lanczosSum;
  gamma *= pow(0.367879, z);
  gamma *= pow(z + 4.15, z + 0.5);
  return gamma;
}
```

Listing 4.1: A Lanczos Gamma approximation with just 4 coefficients is used to calculate directional spreading in a compute shader.

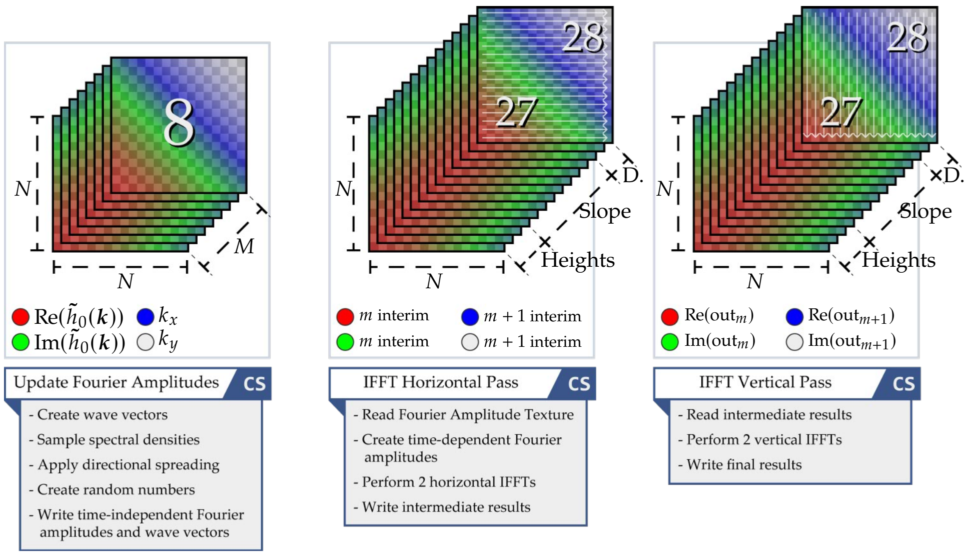

8
N
N
M
● $\mathrm{Re}(\tilde{h}_0(\boldsymbol{k}))$ ● k_x
● $\mathrm{Im}(\tilde{h}_0(\boldsymbol{k}))$ ○ k_y
Update Fourier Amplitudes CS
- Create wave vectors
- Sample spectral densities
- Apply directional spreading
- Create random numbers
- Write time-independent Fourier amplitudes and wave vectors
28
27
N
N
D.
Slope
Heights
● m interim ● $m+1$ interim
● m interim ○ $m+1$ interim
IFFT Horizontal Pass CS
- Read Fourier Amplitude Texture
- Create time-dependent Fourier amplitudes
- Perform 2 horizontal IFFTs
- Write intermediate results
28
27
N
N
D.
Slope
Heights
● $\mathrm{Re}(\mathrm{out}_m)$ ● $\mathrm{Re}(\mathrm{out}_{m+1})$
● $\mathrm{Im}(\mathrm{out}_m)$ ○ $\mathrm{Im}(\mathrm{out}_{m+1})$
IFFT Vertical Pass CS
- Read intermediate results
- Perform 2 vertical IFFTs
- Write final results

Figure 4.4: Time-indepenent fourier amplitudes are updated whenever needed (left). In a first pass (middle) they are used for a inverse FFT applied horizontally. The second vertical pass (right) yields the output: $\mathrm{out}_m \in \left\{h(\boldsymbol{x},t), \nabla h(\boldsymbol{x},t)_x, \nabla h(\boldsymbol{x},t)_y, D_x(\boldsymbol{x},t), D_y(\boldsymbol{x},t)\right\}$

Any random numbers in the remaining process of creating $\tilde{h}_0(\boldsymbol{k})$ (see equation 3.10) need to be generated pseudo-randomly such that they stay fixed on every update. Finally, the time-independent amplitudes are stored in the red and green channel of the band's texture slice and wave vectors in the blue and alpha channel. It has proven to be sufficient using 16 bit per channel.

Two-pass Fast Fourier Transform

A GPU based Stockham FFT is employed that uses shared memory to allow transformation of two signals per thread group. The implementation is part of Unreal Engine and based on the paper [Gov+08]. Govindaraju et al. present several possible algorithms and thoroughly compare them [Gov+08]. Its review is not subject of this work.

The first pass applies the inverse FFT horizontally and the second pass vertically. This produces the three possible outputs of height, slope or displacement for M bands (see figure 4.4). Two texture arrays are required in the process. Their number of slices is chosen according to the number of outputs, taking into account that a single slice can hold the output of two transformed signals.

During the first pass, the time-independent Fourier amplitudes and wave vectors are read and used to calculate the time-dependent amplitudes. Here, the output type needs to be considered to transform them according to equation 3.12, 3.16 or 3.17. A buffer that maps the dispatch's z index to the output type of the two involved bands allows identification. The intermediate results of the horizontal transformation are written to an intermediate texture array. The second pass reads those intermediate results, applies the inverse FFT vertically and writes the final result to a second texture array.

The example depicted in figure 4.4 involves 8 bands. As suggested, the desired output type can be controlled on a per-band basis. For each band both heights and normals are enabled. Thus, 4 texture slices are needed for the height outputs and 8 texture slices for their slope. Only 2 bands have displacement enabled. Therefore, two more texture slices are sufficient. This adds up to 14 $N \times N$ texture slices that are updated every frame. If the imaginary part of the output is not required, it should be dropped during the vertical pass to reduce the amount of texture slices (see suggestions in section 5.1.1).

4.2 Simulating Shallow Water

For the practical use of the LBM, it is of utmost importance to get a deep understanding of the parameters involved. They are directly connected to the stability of the method and essentially define what problem can be simulated. This is even more important for the macroscopic Lattice Boltzmann method since the only adjustable parameters left are the eddy viscosity ν and the lattice size Δx, with everything else written in its dependence.

At the risk of repeating, the model's parametrization is introduced, its implications are elaborated, and it is proposed how to handle and utilize them.

4.2.1 Parameter Clarification

For an eddy viscosity ν and lattice size Δx, both the particle speed e and time step Δt are defined in dependence:

$$e = \frac{6\nu}{\Delta x} \tag{4.18}$$

$$\Delta t = \frac{\Delta x}{e} \tag{4.19}$$

Stability is given when the simulated quantities of depth d and horizontal velocity $\boldsymbol{u}$ satisfy the conditions introduced in section 3.3.1 and 3.3.2:

$$\frac{\sqrt{\boldsymbol{u} \cdot \boldsymbol{u}}}{e} < 1 \tag{4.20}$$

$$\frac{\sqrt{gd}}{e} < 1 \tag{4.21}$$

$$\frac{\sqrt{\boldsymbol{u} \cdot \boldsymbol{u}}}{\sqrt{gd}} < 1 \tag{4.22}$$

$$\sqrt{\boldsymbol{u} \cdot \boldsymbol{u}}\ \frac{\Delta x}{\nu} < 1 \tag{4.23}$$

In terms of practical use, the lattice size Δx translates to the spatial domain in combination with a grid size n to the spatial length $l = \Delta x n$. The real-time requirement prevents arbitrarily increasing n. Thus, for a given grid size n, the lattice Δx should be chosen to span the desired simulation domain.

Equation 4.21 sets a limit on the depth of the simulated water. Since the goal is to employ the shallow water simulation in the context of an ocean simulation supplementing missing features of the FFT approach, it is desired to control the simulated depth to one's liking. Using the particle speed definition 4.18, the stability condition 4.21 and a desired depth d_d that can be simulated, the eddy viscosity ν can be chosen in accordance:

$$\begin{aligned} 1 &> \frac{\sqrt{gd_d}}{e} \\ \Rightarrow \frac{6\nu}{\Delta x} &> \sqrt{gd_d} \\ \Rightarrow \nu &> \frac{\Delta x}{6}\ \sqrt{gd_d} \end{aligned} \tag{4.24}$$

With some basic algebra, it can be found that the stability condition 4.23 is stricter than 4.20. For the velocity this results in:

$$\begin{aligned} \frac{\sqrt{\boldsymbol{u} \cdot \boldsymbol{u}}}{e} &< 1 & \Rightarrow \sqrt{\boldsymbol{u} \cdot \boldsymbol{u}} &< \frac{6\nu}{\Delta x} \\ \sqrt{\boldsymbol{u} \cdot \boldsymbol{u}}\frac{\Delta x}{\nu} &< 1 & \Rightarrow \sqrt{\boldsymbol{u} \cdot \boldsymbol{u}} &< \frac{\nu}{\Delta x} \\ \Rightarrow \sqrt{\boldsymbol{u} \cdot \boldsymbol{u}} &< \frac{\sqrt{gd_d}}{6} \end{aligned} \tag{4.25}$$

In summary, for the simulation to be stable, the simulated water depth d must be less than d_d for stability condition 4.21 to be satisfied, and the velocity $\boldsymbol{u}$ must satisfy both 4.22 and 4.25. The lattice size is chosen to span the desired simulation domain for a given grid size, and the eddy viscosity is chosen to allow simulation of a certain desired depth d_d.

4.2.2 Implications of the Parametrization

While this allows to set up the stable simulation of a certain scenario, it has several implications.

For one, instead of freely choosing the eddy viscosity ν, it is calculated to support the given scenario. It is important to remember that the eddy viscosity is not some value directly used by the model, rather the model depicts a certain eddy viscosity. When the lattice size Δx is doubled, the new smallest vortex the lattice may exhibit is twice of the size. Thus, it is logical that the lattice size Δx is part of the eddy viscosity definition 4.24.

Next, the neighbourhood in the LBM equation is evaluated: $f_i^{eq}(\boldsymbol{x} - \boldsymbol{e}_i \Delta t, t - \Delta t)$. Looking at equation 4.18 and 4.19 it is clear that $\boldsymbol{e}_i \Delta t$ is exactly a neighbouring lattice point. From the implementation's perspective this is very beneficial, as a simple load operation without interpolation can be used. Finally, the time step 4.19 is fixed and depends on the lattice size and the particle speed.

#	l [m]	n	d_d [m]	Δx [m]	ν [m^2 s^{-1}]	e [m s^{-1}]	Δt [s]
1	256	512	4	0.5	0.522	6.2642	0.0798
2	512	512	4	1.0	1.044	6.2642	0.1596
3	1024	512	4	2.0	2.088	6.2642	0.3193
4	4096	512	4	8.0	8.3522	6.2642	1.2771
5	256	512	32	0.5	1.4765	17.718	0.0282
6	512	512	32	1.0	2.953	17.718	0.0564
7	1024	512	32	2.0	5.9059	17.718	0.1129
8	4096	512	32	8.0	23.624	17.718	0.4515

Table 4.1: This comparison of different values for l,n,d_d shows the effect on dependent values for Δx, ν,e,Δt.

With the help of table 4.1, more practical observations can be shown. Lattice size, eddy viscosity and time step grow at the same rate, so doubling the lattice size Δx also doubles the time step. For certain scenarios, the time step Δt gets very small or large. Considering a frame rate of 60 frames per second with a frame time of ~0.16 s, running the simulation with a time step of 1.2771 s as in row 4 means it runs too fast. On the other hand, row 5's time step 0.0282 s would be too slow. Both cases need some closer examination.

Due to the ambitious performance requirements, it is not possible to increase the amount of simulated time steps per frame. Instead, the assumption is made that it is acceptable for the time step to be smaller than the frame time. Intuitively, this can be justified by waves on a puddle moving in slow motion appearing larger than they actually are. Another example is the use of small scale models in film making and even in scientific applications before powerful computers were available.

Clearly, there are less performance concerns about decreasing the amount of simulated time steps per frame. However, it should be avoided to distribute work unevenly throughout frames to prevent introducing spikes in frame time.

4.2.3 Level of Detail Scheme

The presented analysis of the parameters, stability conditions and their implications are now used to formulate of a LOD scheme.

Multiple layers of simulation lattices are stacked on top of each other as depicted in figure 4.5. All are of the same resolution $n \times n$, but the lattice size Δx is doubled for every layer. When centred around a single point in space like the camera position, a large area can be covered while preserving small scale details in camera vicinity. The identical resolution of layers allows for an efficient memory layout, for instance, the usage of texture arrays on the GPU.

To evenly distribute the workload of lower detail layers with high time steps, a sub-stepping approach is introduced. Instead of processing the whole simulation domain just every other frame, it is halved into a sub domain for every lower detail layer and switched through. When the highest level of detail layer is processed in its entirety for every frame, the amount of lattice points being processed per frame converges to $2n^2$ upon increasing the number of layers (see figure 4.5). Obviously, there is a limit on how many layers can be added this way depending on the choice of n.

To formalize this, let o be the index of a LOD layer, where $o = 0$ has the highest level of detail and, therefore, the lowest spatial length. Most importantly, for $o = 0$ the minimum lattice size is defined $\Delta x_0 = \Delta x_{\text{min}}$ and subsequently $\Delta x_{o+1} = 2 * \Delta x_o$. Both the resolution n and the number of sub

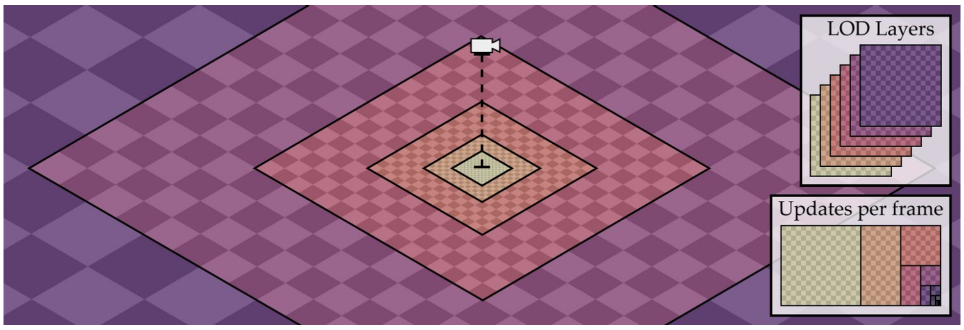


Figure 4.5: Several layers of equal resolution are stacked on top of each other around a centred point. Their spatial side length grows quadratically allowing a large area to be covered. With every layer the simulation domain is halved into increasingly smaller sub domains as shown in the bottom right.

steps M^{sub} of a layer are required to be a power of two. Using a user-defined target or average frame time Δt_f, the number of sub steps M_0^{sub} is the closest non-zero power of two of $\Delta t/(\Delta t_f)$. Thus, time steps less than the frame time are still simulated once per frame, but time steps higher than the frame time are slowed down by using more sub steps. In any case, the number of sub steps for the next lower layer is always doubled $M_{o+1}^{sub} = 2 * M_o^{sub}$ in order to match the timing of layers. An intuitive way to interpret M^{sub} is that a simulation layer is completely simulated after M^{sub} frames passed.

Given a minimum side length of a subdomain $n_{\min} \geq 1$, the maximum amount of sub steps and in turn the highest LOD layer index can be calculated:

$$M_{o_{\max}}^{sub} = (\frac{n}{n_{\min}})^2 \tag{4.26}$$

$$o_{\max} = \left\lfloor \log_2 \left(\frac{M_{o_{\max}}^{sub}}{M_0^{sub}} \right) \right\rfloor \tag{4.27}$$

Table 4.2 shows the values for a specific example. Here, the proposed LOD scheme allows simulating at an lattice size of 0.125 m at the highest detail level and covering an area of 1310.73 km for the lowest detail level.

o	l [m]	Δx [m]	ν [m^2 s^{-1}]	Δt [s]	M^{sub}
0	128	0.125	0.18456	0.01411	1
1	256	0.25	0.36912	0.02822	2
2	512	0.5	0.73824	0.05644	4
3	1024	1.0	1.4765	0.11288	8
4	2048	2.0	2.9530	0.22576	16
5	4096	4.0	5.9059	0.45152	32
6	8192	8.0	11.812	0.90305	64
7	16284	16.0	23.480	1.7951	128
8	32768	32.0	47.247	3.6122	256
9	65536	64.0	94.495	7.2244	512
10	131072	128.0	188.99	14.449	1024

Table 4.2: Listing all layer dependent values for given $n = 1024, n_{\min} = 32, d_d = 16\,\mathrm{m}, \Delta x_0 = 0.125\,\mathrm{m}, \Delta t_f = 0.166\,\mathrm{s}$. With $M_{o_{\max}}^{sub} = 10$, there are a total of 11 layers with a constant lattice speed of $e = 8.8589\,\mathrm{m\,s^{-1}}$.

Layer boundary conditions

The boundary treatment between layers for the proposed arrangement of the layers in figure 4.5. It should be noted that the arrangement is not substantial, and other arrangements could be employed for a different usage scenario (see section 5.2.3 for suggestions).

Boundaries emerge between layers, and the lowest LOD layer has the usual outer boundary that can be found in any finite grid based simulation. Typically, bounce-back, repeating or non-reflecting boundary conditions are employed. LBM with macroscopic variables allows to easily integrate these, and this work uses a very simplistic approach of dampening them towards an idle state. An advantage of the LOD scheme is that this outer simulation domain can be pushed very far away, so that it hardly matters.

Between LOD layers three processes can be observed. First, flow from higher LOD layers may pass to a lower one. This stands in contradiction to the second case of normal flow inside a layer since here a lattice point has two possible sources for neighbouring values. Then, there must be some flow from lower to higher layers.

```
bool TreatOuterLODLayerBoundary(inout int2 xNeighbour, inout int
    layerIndex)
{
  bool treatOuterBoundary = any(xNeighbour < 0 || xNeighbour >=
    TextureSize);
  if (treatOuterBoundary)
  {
    xNeighbour = (xNeighbour + HalfTextureSize) / 2;
    layerIndex++;
  }
  return treatOuterBoundary;
}

bool TreatInnerLODBoundary(in int2 x, inout int2 xNeighbour, inout int
    layerIndex)
{
  int2 lowerBounds = FourthTextureSize / 4;
  int2 upperBounds = 3 * (FourthTextureSize / 4);
  bool isPointOutsideOverlapArea = any(x < lowerBounds || x >= upperBounds
    );
  bool isNeighbourInsideOverlapArea = all(xNeighbour >= lowerBounds &&
    xNeighbour < upperBounds);
  bool treatInnerBoundary = isPointOutsideOverlapArea &&
    isNeighbourInsideOverlapArea;
  if (treatInnerBoundary)
  {
    xNeighbour = 2 * xNeighbour - HalfTextureSize;
    layerIndex--;
  }
  return treatInnerBoundary;
}
```

Listing 4.2: Let `x` be the current lattice point, `xNeighbour` a direct neighbour on the current layer with index `layerIndex` and `TextureSize` equals n. Other variable names are chosen to be self-explanatory.

Outer boundary treatment is required for the flow from low to high LOD layers (see figure 4.6) with two possible approaches. Variables of a neighbouring lattice point can be used directly. This is computationally efficient but an incorrect spatial position is considered. Using interpolation

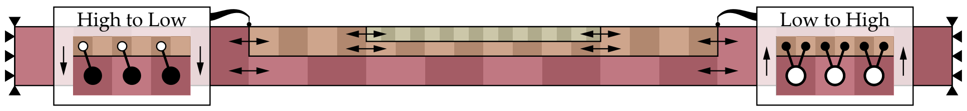


Figure 4.6: A schematic cross section view shows the possible flow between stacked LOD layers and the simulation domain boundary for the lowest detail layer. Two extracts of a close-up top down view depict a possible treatment of flow from high to low layers (left) and from low to high layers (right).

instead, the variables of exact neighbour positions can be approximated. This requires more lattice point evaluations. The first option is depicted on the right in figure 4.6 and used for this work as shown in listing 4.2.

Inner boundary treatment is required where a lower LOD layer has the option to get its neighbouring variables from a higher layer. In this case, there are several options to choose from. First, the neighbourhood variables can simply be retrieved from within the same layer. When instead retrieved from a higher layer, there is an ambiguity because 4 higher lattice points are covered by the lower one. Taking the average or choosing a random point of the 4 candidates would be the most exact solution. From a computational point of view it is simpler to just select the same point every time which is implemented by the code in listing 4.2 and depicted on the right in figure 4.6. Finally, there is the option to combine values from within the layer and from the higher layer.

Adapting the level of detail

So far the described scheme allows depicting different levels of detail but lacks adaptivity. The first step to adaptivity is limiting the number of active LOD layers to M^{act}. Then, the camera's altitude above mean sea level h_{cam} is used to determine the index of the highest active LOD layer o_{act}:

$$o_{\text{act}} = \max\left(0, \left\lceil \log_2\left(1 + \mu\frac{|h_{\text{cam}}|}{l_0}\right)\right\rceil - 2\right) \tag{4.28}$$

Here, μ is a LOD factor that can be used to bias the level of detail towards higher or lower details. Figure 4.7 has the function plotted alongside the graph of the function without rounding that is useful to get the fractional part for fading purposes. In short, the active LOD layer is selected in dependence of the minimum spatial length such that without any bias the first change occurs at three times the minimum spatial length.

When the highest active LOD layer changes, the now obsolete LOD layer is replaced by a new layer of either higher or lower detail. Thus, the number active layers is always fixed. A decrease of altitude means the

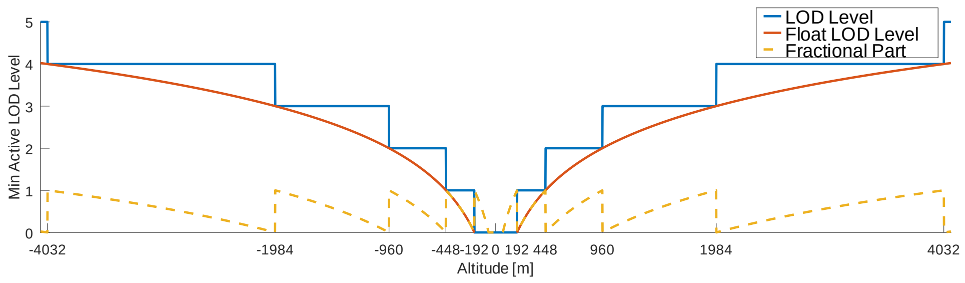


Figure 4.7: Plot of the active highest LOD level for different altitudes. Here $l_0 = 64$ and $\mu = 1$.

newly added layer has higher detail. Here, it can be initialized with the state of the overlapping area of the next lower LOD layer. Upon the increase of altitude, only the inner part of the new layer's simulation domain is currently represented by the next higher layer. In this case, only the inner part is set to the known state while the outer part is initialized to some idle state.

Camera movement parallel to the sea surface needs to be considered as well. The basic idea is to realign each layer once a certain threshold is exceeded. To avoid copying large amounts of memory only the obsolete area of a layer is overwritten and memory access is adjusted by an offset. Figure 4.8 shows an example of a movement and the resulting realignment. This scheme successfully keeps memory changes and the workload during a realignment at a minimum. However, keeping track of multiple offsets, detecting boundaries, copying layer data and integrating sub stepping turned out to be undesirably complicated. Thus, it is not presented in more detail. Instead, in section 5.2.3 suggestions are made on how to alternatively set up an adaptive LOD scheme.

4.2.4 GPU implementation

Simulation is performed in a compute shader, dispatched for each LOD layer's active sub domain, reading from one and writing to another texture array. The procedure can roughly be split in the following steps, for which insights and suggestions are presented thereafter:

1. Load sea bed elevations to shared memory.
2. Calculate the local equilibrium for the centre lattice direction.
3. Iterate the neighbouring lattice points.
    - Treat the LOD boundaries.
    - Calculate the local equilibrium distributions f_i^{eq}.
    - Evaluate the sea bed influence and external forces.

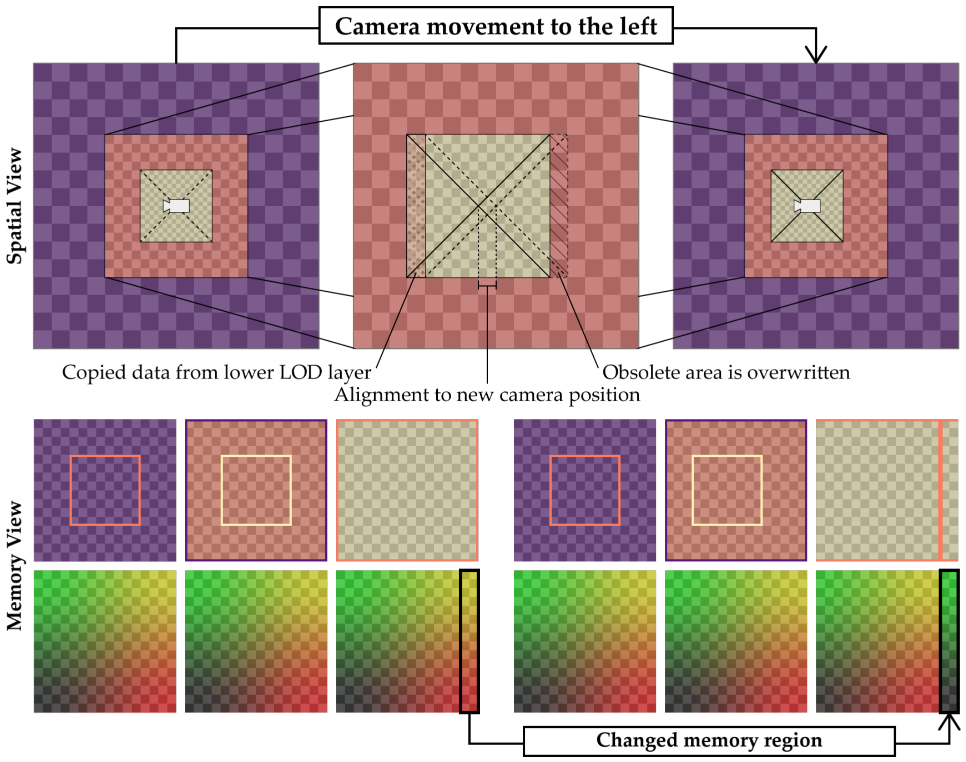

Camera movement to the left
Spatial View
Copied data from lower LOD layer
Alignment to new camera position
Obsolete area is overwritten
Memory View
Changed memory region

Figure 4.8: During camera movement the LOD layers are realigned upon exceeded some threshold. The top left shows the layer arrangement before and the top right after a camera movement to the left. As depicted in the top centre, the change due to realignment of the layer is made up of only a small portion of the layer's area. Reflecting this property to the implementation can be achieved by using an offset every time the memory is accessed. This prevents to move and copy the entire layer's content increases the model's complexity.

4. Get the macroscopic properties depth d and velocity $\boldsymbol{u}$ and ensure stability conditions are satisfied.
5. Calculate the surface normal through central differences.
6. Write the results to an output texture.

Usage of shared memory

In general, the neighbouring access of textures is rather GPU friendly, and the texture holding the simulation values can be accessed with load instead of sample operations. Next to the simulation data, the sea bed elevations need to be loaded or possibly sampled from another texture. Applying the *Peak-Performance-Method* [Bav19] showed that work has to be removed from the texture unit. Storing sea bed elevations in shared memory while loading simulation data in-place yields a better balance of workload and is therefore suggested. Each thread stores the sea bed elevations of its four diagonal neighbours in shared memory. This approach covers all candidates without the need of boundary handling.

Extracting the centre lattice direction

Since the centre lattice point is much simpler (see definition of C_i 3.39), it can be explicitly treated before looping the neighbourhood.

Iterating neighbouring lattice points

The loop over the 8 neighbouring lattice points is explicitly marked with the `[unroll]` keyword. This allows the compiler to unroll the loop and reorder instructions in a way it sees fit, resulting in a gain of performance.

Looking at the equilibrium function shows a lot of divisions:

$$f_i^{eq} = \begin{cases} d\left(1 - \frac{5gd}{6e^2} - \frac{2\boldsymbol{u}\cdot\boldsymbol{u}}{3e^2}\right) & , i = 0 \\ \lambda_i d\left(\frac{gd}{6e^2} + \frac{\boldsymbol{e}_i\cdot\boldsymbol{u}}{3e^2} + \frac{(\boldsymbol{e}_i\cdot\boldsymbol{u})^2}{2e^4} - \frac{\boldsymbol{u}\cdot\boldsymbol{u}}{6e^2}\right) & , i \neq 0 \end{cases}$$

Fortunately, once the simulation is set up the particle speed e never changes, and all divisors can be pre-calculated on the CPU. Henceforth, calculating f_i^{eq} only consists of additions, subtractions and multiplications.

Velocity, depth and stability

When all neighbouring lattice points have been processed, a division by the water depth d is needed to obtain the velocity $\boldsymbol{u}$ (see equation 3.42). A division by zero must be ruled out at this point. If a depth of zero is detected, the velocity is set to zero as well.

Additionally, measures are taken to ensure both depth and velocity are still within their supported limits that keep the simulation stable. In listing 4.3 an implementation is presented to enforce the limits at run-time. Instead of hardly capping the water depth, it is smoothly cut off employing a `smoothmin` function proposed by Quilez [Qui13]. In particular, the current depth d is limited to d_d with a smoothed radius of $0.2 * d_d$. The factor of 0.2 was found to yield satisfying results. While in the best case values never exceed their limits at all, it cannot be ruled out in a dynamic game environment. Without smoothly capping the water height, hard edges are visible which in turn lead to ripples on the water surface. Hardly capping the velocity is not directly visible and in was therefore considered as sufficient.

```
float smoothmin(float a, float b, float k)
{
  float h = max(k - abs(a - b), 0.0) / k;
  return min(a, b) - h * h * k * (1.0 / 4.0);
}

void EnforceStabilityConditions(inout float2 u, inout float d)
{
  d = smoothmin(d, DepthLimit, 0.2 * DepthLimit);
  float magnitudeU = length(u);
  if (magnitudeU > 0.0)
  {
    u /= magnitudeU;
    u *= min(magnitudeU, VelocityLimit);
  }
}
```

Listing 4.3: To ensure a stable simulation, the stability conditions are enforced at runtime. Here, `DepthLimit` equals d_d and `VelocityLimit` is set according to equation 4.25. The `smoothmin` function was proposed by Quilez [Qui13].

Calculating surface normals

Surface normals are calculated with a central difference approach using the sum of depth and seabed of direct neighbours. This is important since the desired normals are those of the sea surface and not of the water's depth. Taking the values during the iteration of neighbours essentially means that normals are created for the last simulation state. However, this saves another neighbourhood evaluation and the performance advantage was considered worth the trade-off.

Writing results to texture

Experiments have shown that 32 bit floating-point precision is required for the depth. Using just 16 bit precision for the depth resulted in the loss of water when run for a longer period of time. Lowering precision per velocity component to 16 bit does not induce a loss of water depth. Using a texture with four 32 bit channels allows to store the depth in a single channel, both velocity components are packed into another single channel and the surface normal in the remaining two channels. For the surface normals this means only 2 of 3 components are stored, the missing third one can be reconstructed using the property that its length is 1.

4.3 Coupling Deep and Shallow Water

Key reason to couple deep water waves obtained from the FFT approach with the shallow water simulation is to get physically-plausible interactions at

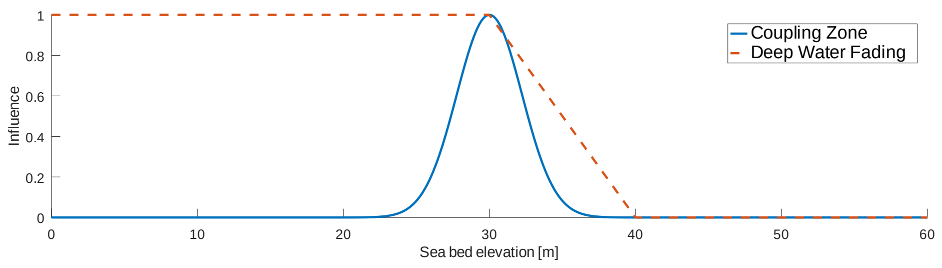


Figure 4.9: Deep water to shallow water coupling zone with $z^{\mathrm{p}} = 30$ and $z^{\mathrm{w}} = 10$. Both peak and width can be used to determine a fading factor for deep water waves.

the inshore and shore. None of the referenced works describe a combination of FFT height field with a physically-based fluid simulation in a suitable way. Either the FFT approach was used to add details to a low resolution simulation [CM10] or the shallow water simulation was coupled with other fluid simulation methods [Oje13a; CMK15].

Therefore, several own ideas of coupling them in a plausible way were tested. One attempt was to derive an external force which can be input to the LBM equation and another was to describe it similarly to the sea bed consideration. Former was considered too complicated in terms of implementation and computational complexity. Latter did not show any advantages over the simple method that was chosen in the end. That is, heights of selected Fourier bands are modified by some weight and used to directly alter the water depth during simulation. The shallow water simulation continues to propagate these changes in subsequent simulation steps.

Weighting the Fourier heights

Finding a suitable weighting is fundamental in this procedure. First, there is a parameter λ_{dw} that globally controls the influence of deep water heights on shallow water. The target frame time Δt_f and the amount of substeps M^{sub} of a LOD layer are considered to normalize the influence for different layers and target frame times. Then deeper water should experience a greater impact than shallow water. Here, the relative depth d/d_d is used. Finally, a bell-shaped function is introduced that allows to control a zone based on the sea bed elevation in which deep water should influence shallow water simulation:

$$\lambda_{\mathrm{zone}}(\boldsymbol{x}) = \exp\left(-\frac{1}{z^{\mathrm{w}}}(z_b(\boldsymbol{x}) - z^{\mathrm{p}})^2\right) \tag{4.29}$$

Figure 4.9 shows the plot of the coupling zone alongside a linear fading factor that is later used to hide certain Fourier bands where shallow water simulation takes over. Putting all introduced weights together, the water depth d is altered by heights of suitable M_{sw} Fourier bands:

$$d' = d + \left(\lambda_{\text{dw}}\, \Delta t_f\, M^{sub}\, \frac{d}{d_d}\, \lambda_{\text{zone}}(\boldsymbol{x})\right) \sum_{m=1}^{M_{sw}} h_m(\boldsymbol{x}, t) \tag{4.30}$$

Selecting suitable Fourier bands

Selecting Fourier bands suitable to influence the shallow water simulation should be done in accordance to the Nyquist-Shannon sampling theorem. For shallow water LOD layers the spatial length, resolution and update rate is known so it is easy to get the sampling frequencies. However, the Fourier bands contain a wide range of frequencies and satisfying the sampling theorem for the highest frequency turned out to be too restrictive. Instead, it is relaxed towards the band's lower frequencies by a user-defined amount. Since amplitudes of higher frequencies are smaller, no visible aliasing is introduced this way.

4.4 Rigid Body Interactions

To efficiently manage rigid bodies a simplified model of physics proxy objects is employed. Collisions are evaluated using simple sphere or capsule collisions and the effect on the shallow water simulation is modelled procedurally. The physics proxy objects are sorted into an octree spatial datastructure on the CPU and an entry is made in a linear GPU buffer. Every frame relevant objects are queried and mapped to thread groups of a GPU compute shader. There, both effects of the proxy objects on the simulation and buoyancy force acting on the proxy object are evaluated. Forces are then read back to the CPU and passed on to the engines physic solver.

Preliminary collision detection

Every frame the octree datastructure holding the proxy objects is queried for intersections with the currently processed simulation sub domains of each LOD layer. A bounding box is created for this purpose, that matches the sub domain's spatial extent horizontally and covers a controllable height vertically. Step 1 in figure 4.10 visualizes this process. For each intersecting proxy object it is then exactly determined in which *subregions* it lies in. A subregion is a subset of the sub domain that can be covered by a single compute shader threadgroup. In step 2 of figure 4.10 the threadgroup is 4×4 where each thread is responsible for one pixel of the shallow water simulation texture. In practice the threadgroup size should be at least 8×8.

A compute shader is dispatched with the required number of thread groups, 4 in case of the third step of figure 4.10. A first buffer holds start

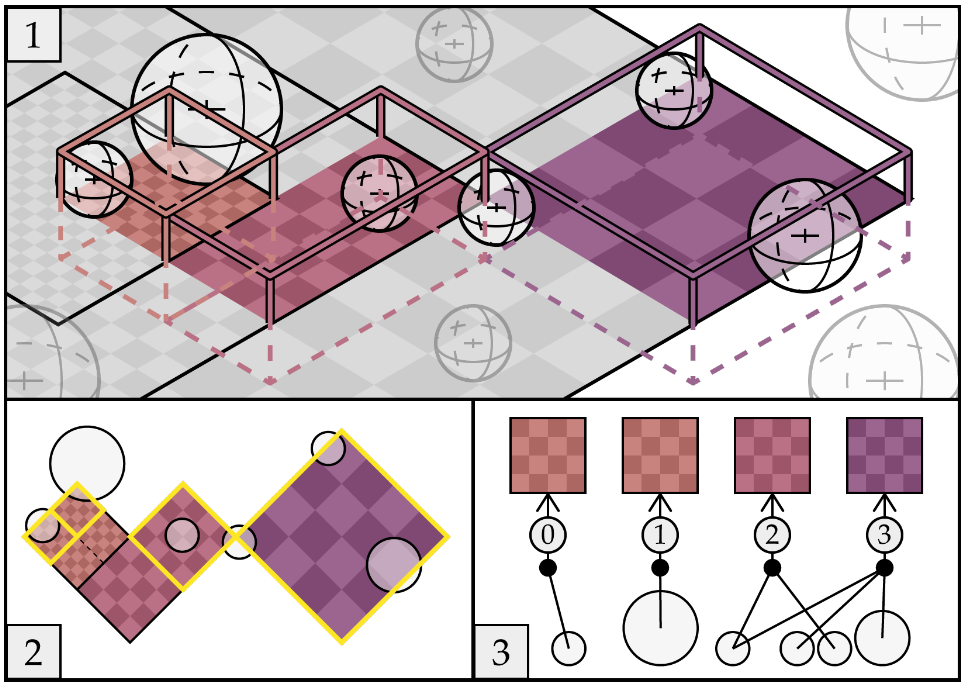


Figure 4.10: The three steps show how objects outside LOD layer sub domains are sorted out and how remaining objects are mapped to thread groups where they are re-identified in order to be processed.

and end indices to a second buffer and it allows each thread group both to identify the simulation texture with the coordinates it is supposed to modify. The second buffer holds the actual indices to the linear physic object buffer that holds all objects. In the linear buffer each object has 64 B of data reserved for its position, its object, collision and effect type and the remaining as free payload that is used depending on the types. Types are identified by checking an associated bit of an 32 bit integer.

Precise collision detection

The reduced number of objects that have to be processed by the GPU make it feasible to iterate the assigned objects. After loading the shallow water depth d, the seabed elevation z_b and the deep water height, the ocean surface's 3D world position $\boldsymbol{w} \in \mathbb{R}^3$ is calculated and a more exact collision test can be performed.

A sphere allows for a very simple collision test. The surface's world position allows to calculate the signed distance to the sphere's surface using the sphere center $\boldsymbol{c}$ and radius r. To prevent branch divergence, processing an object is not aborted if a collision test fails. Instead, any effects are set to zero if the signed distance indicates that the world lies outside the sphere.

In addition to sphere's, capsule are supported as collision geometry. Capsules are convenient, since they can be reduced to the previous sphere collision test. Two points $\boldsymbol{a}, \boldsymbol{b}$ are the start and end of a line. The point on the line $\boldsymbol{w}'$ that is closest to $\boldsymbol{w}$ can be calculated by linear interpolation $\boldsymbol{w}' = \boldsymbol{a} + t * (\boldsymbol{b} - \boldsymbol{a})$ with t [Weind]:

$$t = \frac{(\boldsymbol{a} - \boldsymbol{w}) \cdot (\boldsymbol{b} - \boldsymbol{a})}{(\boldsymbol{b} - \boldsymbol{a}) \cdot (\boldsymbol{b} - \boldsymbol{a})} \tag{4.31}$$

Using $\boldsymbol{w}'$ as the center of a sphere while clamping t in the range of 0 to 1 models the collision with a capsule. Without clamping t the same approach can be used to test with an infinite line.

One reason to use capsules instead of spheres is to prevent missing collisions of fast moving objects or due to large time steps. In this case, increasing the length of the capsule may not yield the most exact collision but it allows to detect them where a simple sphere test would fail.

Procedural effects

To describe the effect of objects on the ocean surface, both depth and velocity of the shallow water simulation are directly manipulated in a procedural way. That means an effect is a function using different parameters to alter the depth or velocity in a certain way. Examples for effects are simply to add or subtract a certain value, applying a sine wave, using random numbers to depict debris causing ripples on the surface, zeroing the velocity to model obstacles reflecting incoming waves or inducing the velocity of a proxy object to the simulation. Attenuating the effects and nullifying them is done using the signed distance to the collision sphere. After all effects are evaluated, the simulations stability conditions are enforced once more.

Buoyancy force read back

Running the simulation completely on the GPU has the difficulty of evaluating buoyancy keeping objects afloat. Reading back data from GPU's memory to be processed by the CPU should be kept at a minimum to prevent saturating the transfer bandwidth. Thus, reading back the simulation data in its entirety is not an option. Instead, the buoyancy force is calculated on the GPU in the same compute shader where effects are applied and written to a buffer alongside a unique id to allow re-identification. To prevent race conditions, a read back proxy object is only assigned to a single thread group where only one thread may write the calculated force. Contents of this buffer are then copied to a staging buffer, that is optimized for CPU read back, once it is detected that a frame was processed by the GPU [Mic18a]. With the unique id the buoyancy force can be mapped to the right game object and is

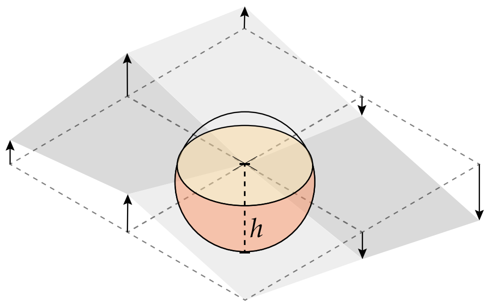


Figure 4.11: Calculating a sphere's submerged volume using the assumption of a flat plane leaves room for error but is simple to compute.

simply forwarded to the engine's physic system. Since it may take a while until this force is updated, it is continuously applied for a set period of time.

Buoyancy is an upward force that acts upon an object immersed in a fluid. Archimedes' principle states that the force equals the displaced fluid's weight. The weight can be calculated using the gravity g, the fluids density $\rho \approx 1000\,\mathrm{kg\,m^{-3}}$ and the volume of the displaced fluid V:

$$\boldsymbol{F}_b = g\rho V \tag{4.32}$$

The main task of buoyancy calculation is to calculate the volume of the dispersed body. Assuming a flat surface, the submerged volume of a sphere with radius r can be determined by calculating the volume of the sphere's one base segment (see figure 4.11) with h as the submerged height [RW04]:

$$V = \frac{\pi}{3} h^2(3r - h) \tag{4.33}$$

$$h = (\boldsymbol{w} - \boldsymbol{c}) \cdot (0, 0, 1)^T + r \tag{4.34}$$

Here, the submerged height has to be clamped to the range $0 \leq h \leq 2r$.

4.5 Output Combination

In the last step both vertices of the ocean surface mesh and a matching texture are prepared. It is done in a compute shader to share values between threads of a group enabling some optimizations. Additionally, the separate preparation step simplifies integration with Unreal Engine's water rendering. The shader involves the steps of calculating world positions according to a projected grid approach, selecting and processing shallow water LOD layers, sampling deep water bands, determining the location of foam, writing results to an output texture and updating the mesh vertices.

Figure 4.12 shows that the compute shader is dispatched to match the output texture resolution. World positions for each pixel are calculated and per group a single vertex of the mesh is updated, such that the output texture directly maps to the mesh.

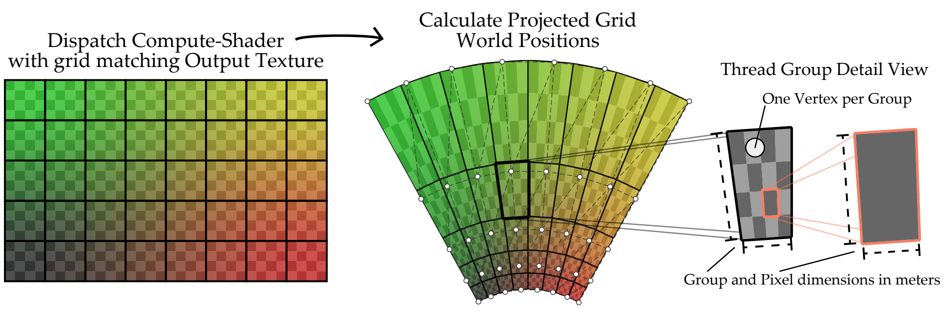


Figure 4.12: Threads of a compute shader are mapped to world positions according to the projected grid approach. Since each thread stands for a pixel in an output texture, the pixel's world space dimensions can be calculated.

4.5.1 Projected Grid World Positions

The attempt to improve the projected grid approach adopts the polar meshing idea suggested by Bowles [Bow13]. Since no further details on the implementation of proposed idea were given in [Bow13], a possible way to implement it is presented here.

A straightforward way to get a spherical mesh is to enforce equidistance of grid points on the same horizontal line to the projector's origin projected on the base seaplane. During bilinear interpolation of the grid corners, the distance between said origin and the interpolated points on lengthwise grid edges can be calculated. The bilinear interpolated grid points can be adjusted to match this distance.

This approach comes with two problems. Interpolated grid points are pushed away from the origin and thus the grid may not cover the whole screen any more. Linear interpolation does not yield same angle's between the lengthwise grid lines.

Snapping the camera's rotation angle to match the grid points during rotation (see figure 3.13) requires the angle between lengthwise grid lines to be equal. Instead of linear interpolation, a form of spherical linear interpolation presented to interpolate normals for shading [HBB03] is used. For two normals $\boldsymbol{N}_a, \boldsymbol{N}_b$ and the tangent $\boldsymbol{N}_t$ orthogonal to $\boldsymbol{N}_a$, the interpolated vector $\boldsymbol{N}(t)$ with $0 \leq t \leq 1$ can be calculated using:

$$\boldsymbol{N}(t) = \boldsymbol{N}_a \cos(t\theta) + \boldsymbol{N}_t \sin(t\theta) \tag{4.35}$$

This can be applied to the projected grid approach, since the four grid corners are calculated and the lengthwise left and right grid edges can be used in place of the normals. Let $\boldsymbol{c}_{xy}$ be a point on the projected grid in homogeneous world space coordinates with $\boldsymbol{c}_{00}, \boldsymbol{c}_{01}, \boldsymbol{c}_{10}, \boldsymbol{c}_{11}$ as the grid's corners. Then angle θ and tangents $\boldsymbol{t}_{00}$, $\boldsymbol{t}_{01}$ to both $\boldsymbol{c}_{00}$ and $\boldsymbol{c}_{01}$ are calculated:

$$\theta = \arccos\big((\boldsymbol{c}_{01} - \boldsymbol{c}_{00}) \cdot (\boldsymbol{c}_{11} - \boldsymbol{c}_{10})\big) \tag{4.36}$$

$$\boldsymbol{t}_{0y} = \frac{1}{\sin(\theta)}\Big(\boldsymbol{c}_{1y} - \cos(\theta)\boldsymbol{c}_{0y}\Big) \tag{4.37}$$

World positions of equal angles can be obtained in three steps. First, linearly interpolate $\boldsymbol{c}_{00}$, $\boldsymbol{c}_{01}$ and tangents $\boldsymbol{t}_{00}$, $\boldsymbol{t}_{01}$ lengthwise with t_y that describes the position along the grid's length:

$$\boldsymbol{c}_{0t_y} = \boldsymbol{c}_{00} + t_y(\boldsymbol{c}_{01} - \boldsymbol{c}_{00}) \tag{4.38}$$

$$\boldsymbol{t}_{0t_y} = \boldsymbol{t}_{00} + t_y(\boldsymbol{t}_{01} - \boldsymbol{t}_{00}) \tag{4.39}$$

Use spherical linear interpolation 4.35 with t_x describing the position along the grid's width:

$$\boldsymbol{c}_{t_x t_y} = \boldsymbol{c}_{0t_y} * \cos(t_x * \theta) + \boldsymbol{t}_{0t_y} * \sin(t_x * \theta) \tag{4.40}$$

After perspective division, this yields the standard projected grid mesh with equal angles. The grid point distance normalization as described previously still has to be applied.

To tackle the problem of pushing vertices into the screen, the distance normalization can be modified. Instead of pushing away centre grid points near the camera, points to the edges should be pulled towards the projector's origin. Naturally this can be achieved by instead using the distance to the centre grid point during normalization. As it can be seen in figure 3.13, the grid corners span an arc on the base sea plane around the projector's origin projected on the plane. Thus, the distance to the centre point equals the distance to the grid corner minus the arc's sagitta.

4.5.2 Combining Shallow Water Detail Layers

With the world positions ready, the next step is to select appropriate shallow water simulation LOD layers (see figure 4.13). For this purpose, each LOD layer has a scale and translation that allows to convert world positions to texture *uv*-coordinates. Starting with the highest detail layer, world positions are converted to *uv*-coordinates until two layers are found for which the coordinates lie within the valid range of 0 to 1. If none of the coordinates match this criterion this means the world position is not covered by the shallow water simulation and for depth, velocity and surface normal, a fall-back value is used.

Searching for two layers allows to fade the boundaries of layers as depicted in figure 4.13. The *uv*-coordinates are used to decide where fading needs to be applied. Additionally, the highest detail layer is faded according to the fractional part of altitude LOD level selection as shown in figure 4.7.

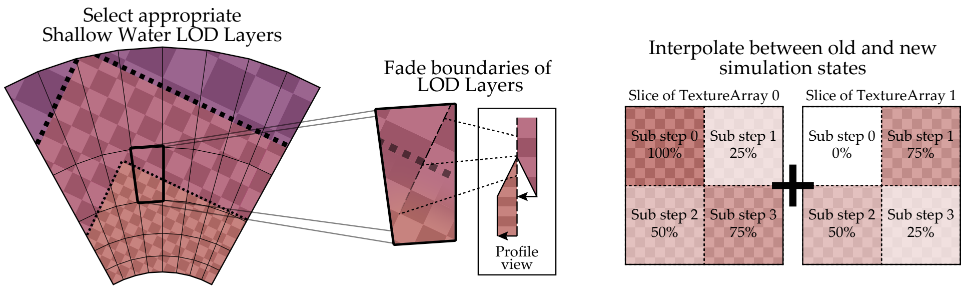


Figure 4.13: The arrangement of multiple LOD layers in world space on the projected grid is shown in the left. Their boundaries are faded to hide discontinuities of surface normals. Long time steps are hidden by interpolating between old and new simulation states.

The presented shallow water simulation reads the previous state from one texture, writes the result to another and swaps read and write textures once all sub steps have been processed. Since the time step is fixed and can get rather high for low detail LOD layers or certain set-ups, an interpolation between old and new simulation states is employed to hide transitions. Due to the sub stepping approach, old and new states may reside in different textures for a single layer and each sub domain needs to be interpolated individually. Thus, following steps are taken to interpolate with respect to sub steps:

1. Sample depth, velocity and surface normal from both textures of the LOD layer.

2. Decide which texture holds the old and which the new simulation state. Let $\boldsymbol{uv} = \left(uv_x, uv_y\right)$ be the layer's uv-coordinates, M_x^{sub}, M_y^{sub} the number of substeps in x and y direction respectively and i_{cur} the currently active sub step. Then the sub step of the target $\boldsymbol{uv}$ can be calculated with:

$$i_{\text{tar}} = \left\lfloor uv_y * M_y^{sub} \right\rfloor M_x^{sub} + \left\lfloor uv_x * M_x^{sub} \right\rfloor \tag{4.41}$$

   If the target sub step i_{tar} is smaller than the current sub step i_{cur}, it has already been updated and the the sample from the texture that is currently written to by the simulation holds the new value, and vice versa.

3. Calculate t used for interpolation with:

$$t = \begin{cases} 1/(M^{sub})(i_{\text{cur}} - i_{\text{tar}}), & \text{if } i_{\text{tar}} < i_{\text{cur}} \\ 1 + 1/(M^{sub})(i_{\text{cur}} - i_{\text{tar}}), & \text{if } i_{\text{tar}} \geq i_{\text{cur}} \end{cases} \tag{4.42}$$

4. Interpolate depth, velocity and normal from old to new using the interpolation value t as depicted in figure 4.13.

4.5.3 Combining Deep Water Bands

For the combination of deep water all heights, normals and displacements obtained from the FFT are iterated. Here, two optimizations using shared memory are employed. Both the maximum water depth of the shallow water simulation and the minimum pixel dimensions (see figure 4.12) of the group are determined. In HLSL, minimum and maximum atomic operations on shared memory are only possible for unsigned integers. Since both water depth and pixel dimension are not negative, their bit pattern can be interpreted as an unsigned integer `uint` to determine the minimum and maximum with atomic operations. The unsigned integer result in shared memory can then be re-interpreted as `float`.

If the maximum water depth of a group is smaller than some epsilon, the iteration of Fourier Transform outputs can be skipped completely. The group's minimum pixel dimension can be used to skip processing single bands, depending on their wavelengths. Just like when selecting bands for coupling with the shallow water simulation in 4.3, taking a band's minimum wavelength is very restrictive. So it is again relaxed towards the band's maximum wavelength by a user-defined amount. If this acceptable wavelength is less than twice the minimum pixel dimension, the band is skipped. In both cases it is important to let the compiler know to branch the if statements using the `[branch]` parameter, otherwise they may be evaluated nonetheless.

To prevent hard transitions due to skipping bands, their output is faded by weighting it by the result of the acceptable wave length minus the current pixel's dimension clamped to the range of 0 to 1. Additionally, a weight as depicted in figure 4.9 is applied to bands of high wavelengths. Low wavelengths bands are kept everywhere to keep small scale details on the otherwise little detailed shallow water surface.

Since heights are used for offsetting the mesh, their condition to be included is based on the group's dimension instead of a pixel (see figure 4.13). This is similar to the approach suggested by Bruneton, Neyret, and Holzschuch [BNH10], although here no BRDF representation is used.

4.5.4 Foam Generation

As introduced in section 3.2.7, the Jacobian determinant of the displacement transformation can be used to detect locations of breaking waves, spray or foam. It requires the calculation of the displacements partial derivative in both x and y direction (see equation 3.18) which is also used in equation 3.19 to apply the slope correction.

Calculating the partial derivative is implemented using the central differences approach. This comes at the cost of four texture samples per displacement direction and therefore totals in eight samples per band with enabled displacement feature.

Foam is expressed as a value between 0 and 1, where 0 means no foam and 1 means maximum foam. To control the amount of foam using the Jacobian determinant, two parameters suggested by Tcheblokov [Tch15] are adopted. They allow to specify a threshold at which foam starts and to control the foam's strength or thickness. Looking at figure 3.8 and at the formula of the following HLSL code, it can be seen that foam is located at crests of choppy waves:

```
float foam = saturate(FoamIntensity * (-JacobianDet + FoamThreshold))
```

Listing 4.4: Foam formula from [Tch15].

To generate foam for shallow water various approaches were tested and the ones that proved to be useful have similarity with the suggestions to detect breaking waves by Fournier and Reeves. They find that the surface's curvature needs to be high and the particle speed needs to be faster than the phase speed [FR86]. Taking the dot product of the velocity $\boldsymbol{u}$ and the xy components of the surface normal foam can be placed on the face of steep waves. The relative velocity magnitude determined in relation to the velocity magnitude limit adds foam in areas of high velocity. Combined using the maximum of both and multiplied by a shallow water foam intensity factor λ_{swf}, the result is subtracted from the Jacobian determinant. Thus, both deep and shallow water foam are combined into one single description.

$$swfoam = \lambda_{swf} \max\left(\boldsymbol{u} \cdot \boldsymbol{n}_{xy}, \frac{|\boldsymbol{u}|}{1/6\sqrt{gd_d}}\right) \tag{4.43}$$

4.5.5 Shoreline Mesh Correction

Determining the water surface for the shallow water simulation requires to add the simulated water depth and the sea bed elevation which simply is the terrain. Here, the problem is that for a water depth of zero, the generated surface matches the terrain. Rendering both the terrain and the water surface results either in z-fighting or overlapping polygons where the terrain's meshing differs from the ocean surface. Figure 4.14 depicts the manifestation of the problem in vicinity to the shore where it is particularly noticeable. To some extent the problem is connected to the projected grid approach, since the camera movement results in different sampling locations.

Two measures are taken to tackle the problem. First, the height to define the vertex' z-coordinate of a group (see figure 4.12) is set to the group's minimum water height. Again this is made possible with atomic operations on shared memory. This measure prevents triangles in direct vicinity to the shore to overlap the terrain as it can be seen in figure 4.14. Additionally, it reduces the projected grid's swimming artefact. To prevent the ocean mesh to clip with the terrain mesh, the idea is to push the ocean's mesh vertices

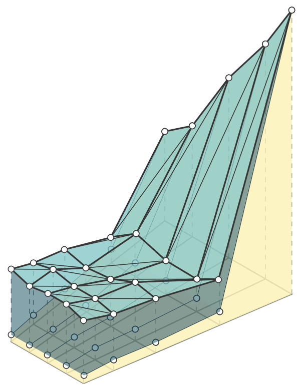
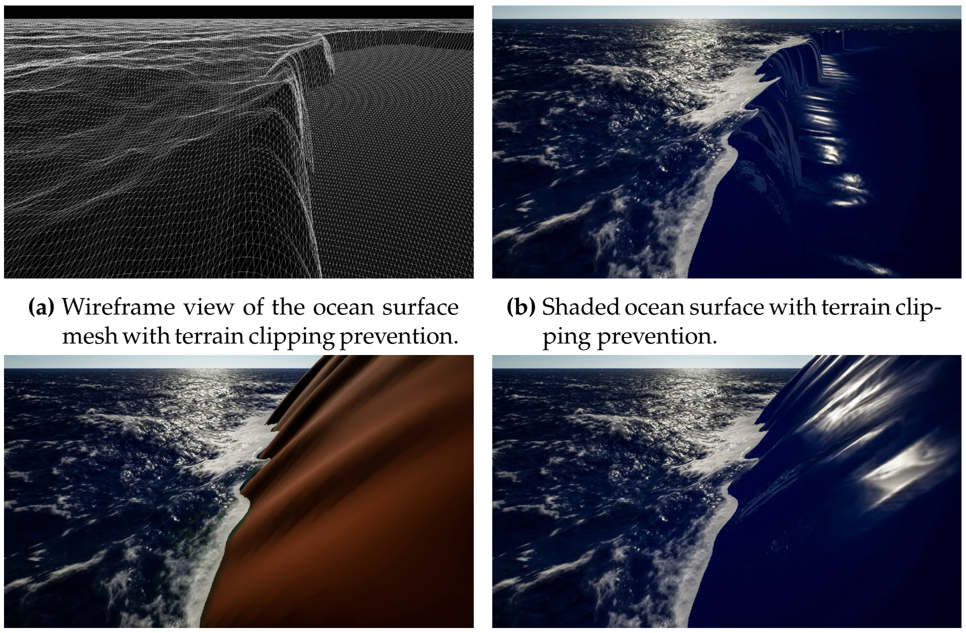


Figure 4.14: Without precautions the water surface mesh covers the terrain in direct vicinity to the shoreline. Additionally, its vertices match the locations of the terrain where water depth is zero.



(a) Wireframe view of the ocean surface mesh with terrain clipping prevention.

(b) Shaded ocean surface with terrain clipping prevention.

(c) Shaded ocean surface with terrain clipping prevention and visible terrain.

(d) Shaded ocean surface without terrain clipping prevention.

Figure 4.15: Different views show the terrain clipping prevention in comparison to the ocean surface without clipping prevention.

below the terrain where the depth d is below a certain threshold. This needs to be done smoothly or the problem of figure 4.14 happens in the opposite direction. Thus, vertices are pushed further below the terrain the higher they are above the average sea level. In figure 4.15 the result of the suggested clipping prevention is presented alongside the ocean surface matching the terrain when it is not applied.

5 Results and discussion

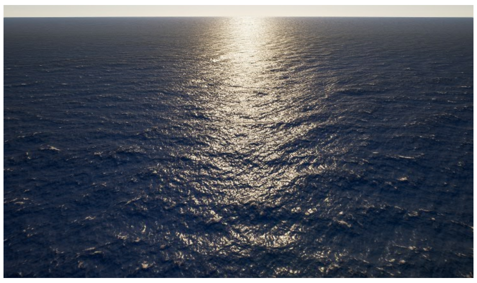
All performance tests were run in the UE 4.25 at a resolution of 1920×1080 on a system with an *Intel® Core™i7-9700F* processor, a *NVIDIA GeForce RTX™2070 Super* graphics card, 32 GB RAM and Windows 10 OS. All measurements of timing and additional presented metrics were obtained using *NVIDIA Nsight Graphics 2020.3.1*. Here, two frames were recorded for each test case, and the better result was used.

5.1 Deep Water Simulation Results

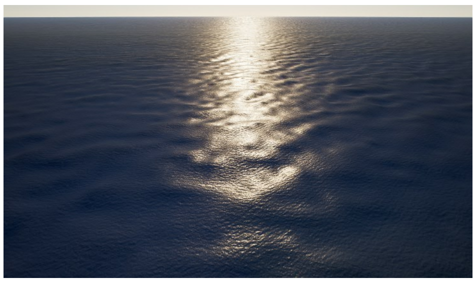
The Fast Fourier Transform approach allows generating a realistic animated surface of the open ocean that was already employed in cinema. Sea states during different weather conditions can be reconstructed using wave spectra [Tes01] that were created using real life measurements of the ocean. Combining swells and wind seas can be used to depict mixed sea, a realistic sea that is used in maritime training simulators [PP20]. Figure 5.1 shows four different sea states generated at the same absolute time $t = 0$. Depicting a wide range of sea states and changing them seamlessly at runtime by using fixed sampling points and random numbers, renders the approach very flexible and interesting for the use in a game environment. Several video games adopted the approach [Tch15; Ang+18; MT19] to different extents, where the latest work [MT19] also allows modelling swells and local wind sea. Both their result in figure 5.1e and this work's results show low frequency swell waves and high frequency wind waves travelling in different directions.

5.1.1 Performance Analysis

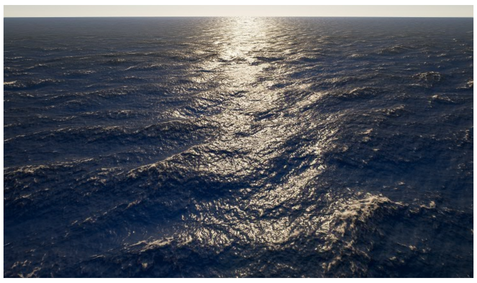
While the FFT takes an absolute time as an input and offers complete freedom in terms of temporal adaptivity, it was run at full frame rate during the following performance tests. Six different quality settings allow examining the impact of different resolutions $N \times N$ and amounts of performed FFT's. The results are listed in table 5.1. Both the size and update time of the time-independent amplitudes $\tilde{h}_0$ depend on the number of employed frequency



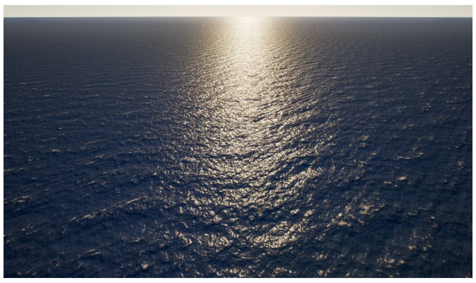
(a) Sea with swell and local wind.



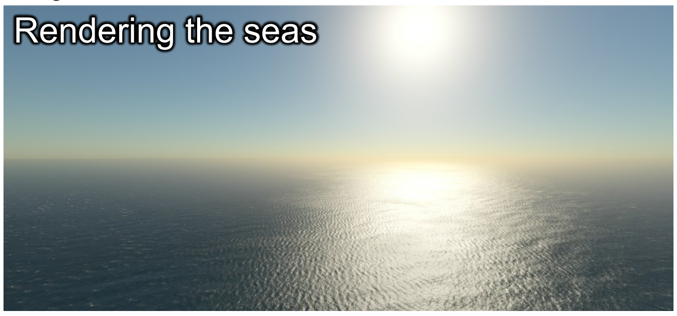
(b) Sea with swell and low local wind.



(c) Sea with huge swell and local wind.



(d) Sea without swell and local wind.



(e) Image of the rendered sea with advanced lighting in the distance [MT19].

Figure 5.1: Different ocean sea states of the current work in comparison to a similar approach from [MT19].

bands and the resolution. Since time does not scale at the same ratio at which the texture increases, it is a hint that the GPU is not able to work at full capacity for little workload. Nevertheless, the execution times of 0.0143 ms to 0.0410 ms are reasonably low to call the movement of spectrum sampling to the GPU a valid option. As updating these amplitudes only takes place when a sea system has changed, the GPU implementation is only advisable when they change at runtime. To make a definite statement regarding this matter, a comparison with a CPU variant should be conducted in the future. However, it was shown that the generation of the time-independent amplitudes $\tilde{h}_0$ is suitable for parallel implementation and a solution to evaluate the gamma function on the GPU has been proposed in listing 4.1.

Timing of the Inverse Fast Fourier Transform (IFFT) again shows that

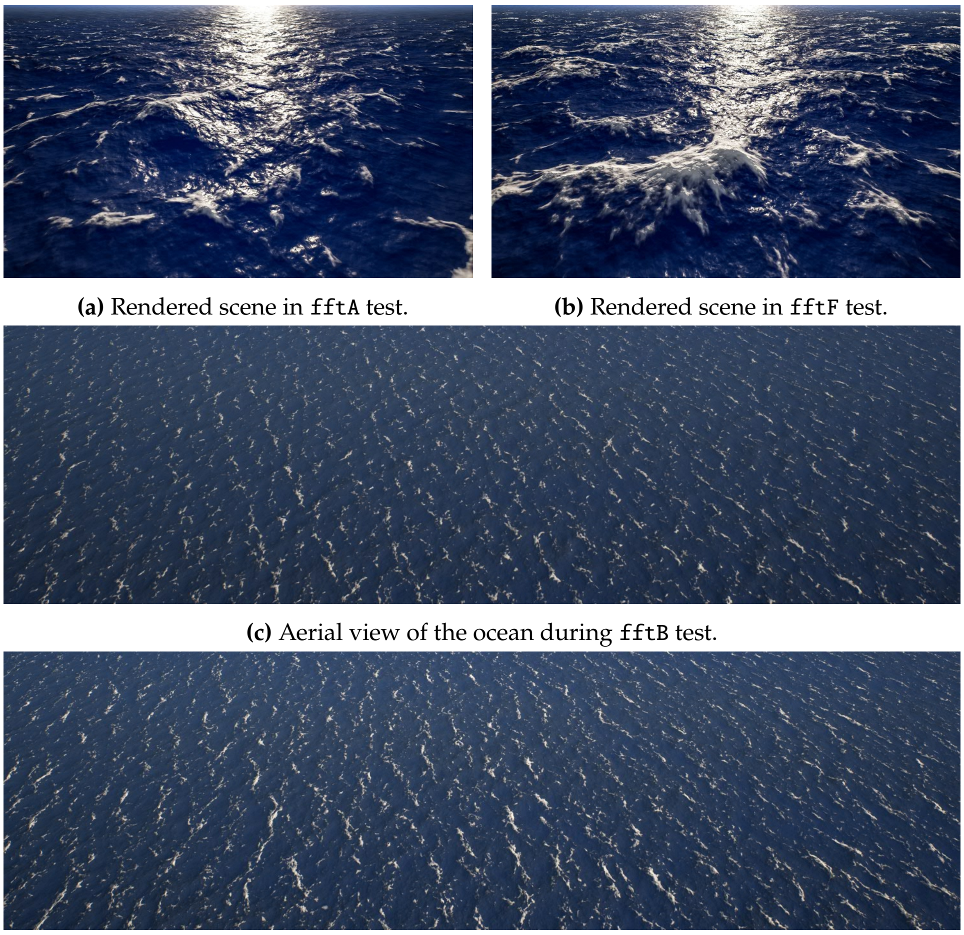
Name of the test	`fftA`	`fftB`	`fftC`	`fftD`	`fftE`	`fftF`
Set up parameters						
Fourier Size N	64	64	64	256	256	256
# Fourier Transforms	12	24	30	12	24	30
# FFT normals	0	6	6	0	6	6
# Central difference normals	6	0	0	6	0	0
# FFT displacements	3	3	6	3	3	6
Stats						
$\tilde{h}_0$-Amplitude-Texture size (kB)	192	192	192	3072	3072	3072
FFT-Texture size (kB)	192	384	512	3072	6144	7680
$\tilde{h}_0$-Amplitude Update Time (ms)	0.0143	0.0143	0.0143	0.0410	0.0402	0.0410
IFFT First Pass Time (ms)	0.0161	0.0220	0.0230	0.0397	0.0596	0.0701
IFFT Second Pass Time (ms)	0.0102	0.0115	0.0123	0.0305	0.0489	0.0581
IFFT Total Time (ms)	0.0263	0.0335	0.0353	0.0702	0.1085	0.1282
Output Combination Pass Time (ms)	0.6659	0.8174	1.0332	0.7355	0.9380	1.1996
UE Water Rendering Time (ms)	1.3279	1.3235	1.3187	1.3374	1.3478	1.3542

Table 5.1: List of six different quality settings and their time measurements. For all quality settings there are two swell systems with 1 band each and one wind sea system with 4 bands.

an increase of resolution does not translate to a proportional increase of execution time. The same is noticeable upon increasing the number of Fourier transforms. Here, doubling the number of FFTs translates to an increased timing of 1.27× for $N = 64$ and 1.54× for $N = 256$. The first pass taking a bit longer than the second pass can be explained by the fact, that the inputs are modified to yield the desired output of height, slope or displacement. The highest execution time of 0.1282 ms for 30 transforms at $N = 256$ is clearly fast enough for real-time usage. To put this result into perspective, a single FFT with $N = 256$ performed on an *Intel® Core™2* processor in 2012 by LeBlanc et al. took 17.2 ms [LeB+12].

The pass for the combination of the simulation outputs and sampling of the Fourier outputs (*Output Combination Pass*) shows much higher execution times ranging from 0.6659 ms to 1.1996 ms. An increase of the resolution and the number of FFT transforms has a higher absolute impact on the execution time. This indicates that once a fast GPU-based FFT is in place, the main task is to use their outputs efficiently.

Comparing `fftA` with `fftB` shows that using a central difference approach to calculate the surface normals is faster than using normals created with the FFT approach. Reason for this is not because of the additional cost of the FFTs. This was the suggested consideration to take into account when using FFT-based normals in the past [Tes01; LeB+12]. Instead, the additional amount of texture sampling reduces the texture throughput (*L1TEX Throughput*) from 81.6% in `fftA` to 58.2% in `fftB`. When calculating central differences in `fftA`, texture data in direct neighbourhood is taken



(a) Rendered scene in `fftA` test. **(b)** Rendered scene in `fftF` test.

(c) Aerial view of the ocean during `fftB` test.

(d) Aerial view of the ocean during `fftE` test.

Figure 5.2: Rendering of the scene as taken during quality tests and additional aerial views.

into account which was already loaded in the cache. For FFT-based normals a different part of global memory has to be accessed, introducing a latency slowing the execution.

Looking at the memory layout of the final FFT outputs in figure 4.4 shows, that half of it is used to store imaginary values that are not required. They are only required in the intermediate step and therefore should be dropped to effectively halve the texture size. Intuitively, this would allow running `fftB` at the time of `fftA`.

A visual comparison of figure 5.2a and 5.2b allows the conclusion, that both central difference normals and FFT-based normals produce a similar and pleasant look. Tessendorf suggested, that the central difference approach is less suited to depict small wavelengths depending on the finite difference [Tes01]. With the band separated approach, a suitable difference for the neighbourhood evaluation can be chosen for each band. Thus, normals of both high and small wavelength waves can be well represented.

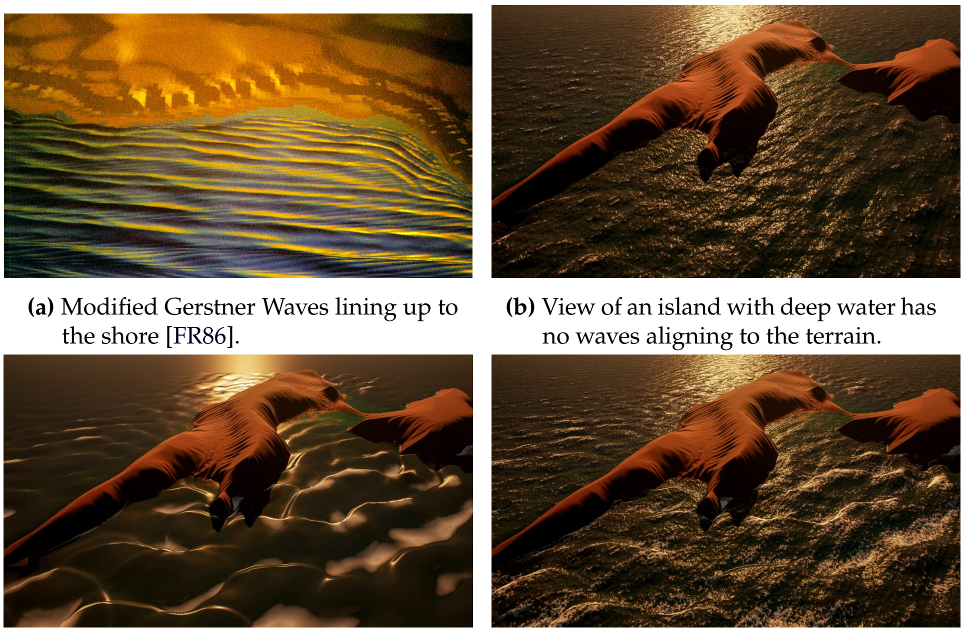
Since the amount of different textures sampled has such a high impact on performance, test `fftE` was repeated without skipping bands of unacceptable wavelengths (as introduced in section 4.5.3) by marking the if statement with the `[flatten]` parameter. In this case the execution time increased from 0.9380 ms to 1.1139 ms what can be attributed to a higher amount of both instructions and texture samples. The ability to discard a high amount of work for whole thread groups shows the benefit of using a compute shader to combine the outputs.

Increasing the resolution N is desirable as it increases the spatial length of individual bands and therefore decreases the visibility of their tiling nature. This is particularly noticeable from the aerial perspective as depicted in figure 5.2c and 5.2d. From the performance point of view it differs in that the GPU's L2 cache is more heavily utilized and has a lower hit rate. This is expected since more data needs to be loaded from global memory. On average the *L2* throughput is increased by 8.2 percentage points from 20.4% in `fftB` to 28.6% in `fftE`. For $N = 256$ the load on L2 varies heavily reaching up to about 75%. This may stem from the fact that different thread groups read very different slices of the Fourier texture array. Besides reducing the amount of texture data it should therefore be considered to sort the texture slices by their wavelength to increase locality. This has the additional benefit, that once one Fourier output has an unacceptable wavelength, all the following can be rejected.

To close this performance analysis, it can be said that the main concern of the FFT approach shifts from the transformation itself to the efficient utilization of its outputs. Their amount and arrangement in memory is of utmost importance and any chance to reduce the amount of evaluated texture samples should be taken. Due to time limitations, the suggested improvements of the memory layout could not be implemented.

5.2 Shallow Water Simulation Results

The goal of an additional fluid simulation is to depict several features that are not possible with the presented FFT approach. Figure 5.3b shows an island from the bird's eye view, rendered using only the FFT approach. Here, the ocean shows no awareness of the land. The same scene rendered only with the shallow water simulation in figure 5.3c shows waves, aligning to the shore similar to Fournier and Reeves' work shown in figure 5.3a. Together with the FFT approach, lacking details are reintroduced as shown in figure 5.3d. Thus, the benefits of the FFT approach are supplemented with waves that converge or diverge based on the topology, diffract due to obstacles and change their speed, when they run up the shore according to



(a) Modified Gerstner Waves lining up to the shore [FR86].

(b) View of an island with deep water has no waves aligning to the terrain.

(c) Ocean surface as created by shallow water simulation lacks detail.

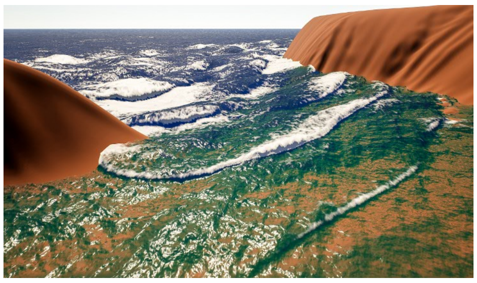
(d) The combination of shallow and deep water has waves lining up the shore and rich surface detail.

Figure 5.3: Comparison of individual deep and shallow water parts and its combination to the ocean surface generated by Fournier and Reeves in 1986.

the shallow water dispersion relation.

A closer look in figure 5.4a shows that the coupling zone and deep water fading (see section 4.9) emphasize the waves in this transition zone too much. Depending on the simulated viscosity that in turn depends on lattice size and the stably supported depth, waves may run out too soon before reaching the shore as expected. Here, a better way to control where the coupling happens needs to be introduced. One way could be to introduce an artist controlled map to allow manual placement of coupling zones instead of simply calculating it from the terrain height. Deep water fading could still be derived from this coupling map, but it needs to be improved to prevent exaggerating waves in the coupling zone.

Additionally, the coupling does not represent small waves well. Since the FFT wave heights directly modify the shallow water state, it takes some time for them to develop their effect. Weighting them more strongly will induce too much of a wave to the shallow water simulation over time. Therefore, the weight has to be tuned down which does not give small waves enough time to develop a meaningful effect. Having a higher weight for bands of small wavelengths could remedy this. It was not pursued further due to time limitations.


(a) In the deep to shallow water transfer zone, waves are weighted too strongly.

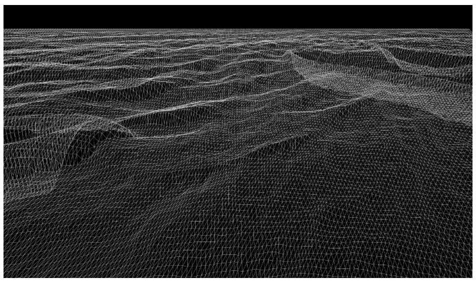

(b) The surface mesh has low resolution on the face of steep waves.

Figure 5.4: This view on the ocean from a bay shows diffracting waves.

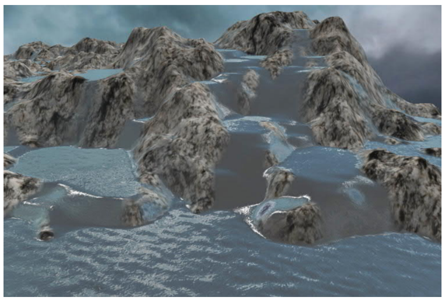
Calculated foam for shallow water has low detail as it can be seen in figure 5.3c. The combination with the deep water Jacobian foam allows to increase its intensity where foam would arise according to shallow water waves. Since this detection mostly matches the steep face of the wave, the shallow water foam intensity needs to be set rather high in order to show it on top of the waves, where one would expect it. This has the side effect of creating foam at places where it is not supposed to be.

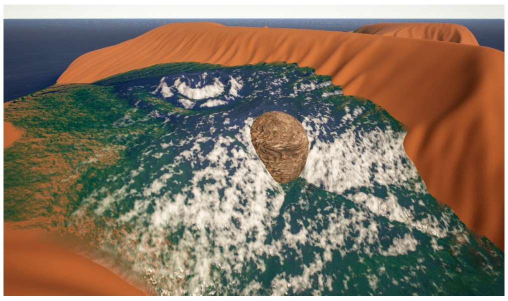
A better way to handle foam would be to detect its location and then create and dissipate it over time. This is described by Tcheblokov for the FFT approach. In addition to this it could be advected using the shallow water velocity field for an even more realistic look. Improving the foam and adding other missing features, i.e. spray or breaking waves, is out of the scope of this work but should be considered in the future.

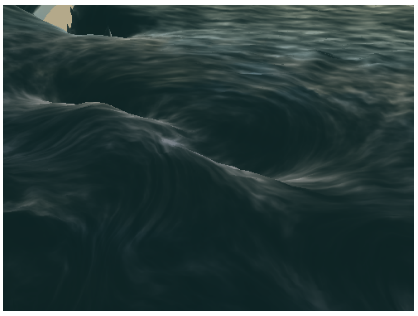
5.2.1 Comparison of Simulation Methods

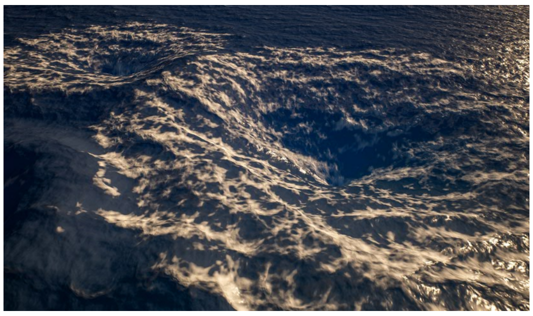
The surface of just the shallow water simulation in figure 5.3c is very dull and lacks fine grain details. For one, this is because of a rather low lattice size Δx of the LOD layers at the depicted altitude. Simply using lower lattice sizes is not possible because the simulation domain either becomes very small or the grid resolution unmaintainable high. This is true for all integration schemes, although the relationship between the type of flow that can be stably simulated and its implications on the time step are different. Here, a short comparison with other simulation methods is presented,

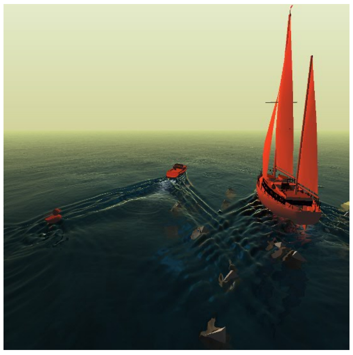
The macroscopic LBM is limited to subcritical flow and a low Lattice Reynolds numbers according to its stability conditions, independent of the employed lattice size. Other methods like the pipe model, the proposed model by Chentanez and Müller and also the standard LBM are not necessarily that much restricted. A low Lattice Reynolds number means, that the flow is dominated by viscous forces and does not show turbulent behaviour. In the scene of figure 5.5b and 5.5d, the water surface therefore


(a) Water runs down a hill [Kel17].


(b) Water runs down a hill and around rock.


(c) Flow of an eddy becomes visible due to advected surface details [CM10].


(d) Without advection the flow of an eddy cannot be seen.


(e) Clearly visible waves of boats [**2009Coords**].

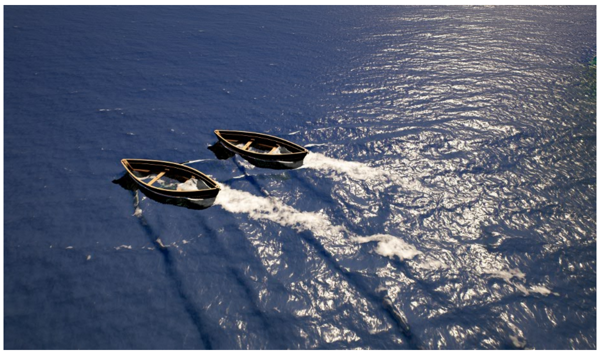

(f) Soft waves due to high an eddy viscosity.

Figure 5.5: Comparison of various scenes from this work and related work.

settles in a steady state of laminar flow. The only option to stably simulate a more detailed surface is to increase the number of lattices [Moh19, p. 111]. But adding external forces to the fluid due to effects of moving rigid bodies or the coupling with FFT heights can hide the laminar nature of the flow and prevents it to come to a steady state. Chentanez and Müller solve the problem of little surface details by advecting texture coordinates with the fluid velocity to visualize the flow as depicted in figure 5.5c. This is not included in this work but should be considered in the future as it would greatly enhance the feeling of a moving water surface, even during laminar flow.

Setting up the simulation to support a desired depth allows to control the simulation intuitively. To stably simulate rather high water depths, the depicted viscosity is increased as shown in section 4.2.2. This makes it difficult to simulate smaller features like waves of a boat (see figure 5.5f) in a pronounced way. Figure 5.5e shows boat waves that are created by solving the wave equations especially for the purpose of object induced waves [CS09]. Therefore, a trade-off has to be made whether to favour smaller features by decreasing the desired depth or instead to favour waves that react to the seabed even in deeper areas.

In contrast to the standard LBM, the key advantages are the reduced memory requirement and the direct use of physical variables. Instead of storing 9 probabilities per lattice point only the depth and velocity are stored. On the other hand, for the standard LBM many improvements were introduced to allow stable simulation of more turbulent flows, although being more complicated and less computationally efficient [Zho19a].

Both the macroscopic LBM for shallow water simulation and the virtual pipe model are suitable for a parallel GPU implementation. With the reduced memory requirement of the reformulated LBM a substantial drawback is removed. In contrast to the pipe model, the shallow water equations include the velocity self-advection and it is not required to calculate the velocity in an additional step. While more precise stability conditions for the macroscopic LBM are advantageous, they are very limiting on the type of flow. Here, a more in-depth comparison of both methods would be an exciting research for the future.

5.2.2 Performance Analysis

Performance of the LBM shallow water simulation and the merging of layers to the output texture is evaluated for different quality settings including a minimal reference setting listed in table 5.2. All tests have 5 active layers at the same time with identical minimum lattice size, but they differ in resolution and the highest active LOD layer. Figure 5.6 shows an image of the test scene for both `sweA` and `sweE` alongside a debug view that shows LOD layer selection and fading.

Both grid resolution n and number of active LOD layers dictate the size of the texture. Since four channels of 32 bit each are used, the memory usage reaches ~ 327 MB for `sweE` and `sweF`. One channel is dedicated for the water depth, one for both velocity components and two for the surface normal. This attempt to keep all data in a single texture has several drawbacks. It is not necessary to store components of a normal with 32 bit precision. Packing two velocity components into a single channel involves some bitwise operations that prevent using texture interpolation for them. During simulation no

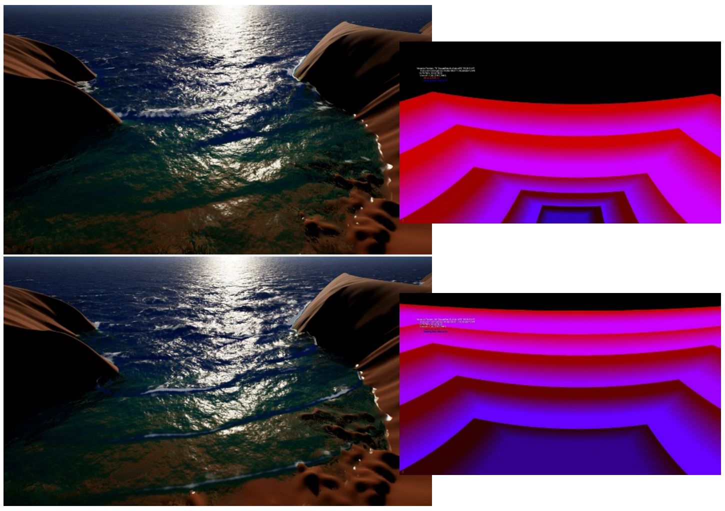


Figure 5.6: View of the scene during test and alongside of a debug output that shows LOD layers and fading as they are mapped on the surface mesh.

interpolation is required, but deriving foam information with point sampled velocities produces block-like foam. To summarize this, it was found that water depth needs 32 bit precision but trying to pack depths, velocity and surface normals into one single texture is not advisable. Instead, a second texture just for the surface normal could be used. Another idea that could be tested is to keep two different sets of the simulated physical properties. One for simulation purposes using high precision and one for displaying the results using low precision.

The LOD system has two major benefits. It allows covering a large area, and areas of low detail can be used as the initial state for higher detail areas. With the addition of sub stepping the amount of simulated lattice cells per frame is evenly distributed. Both `sweA` and `sweD` process the same amount of cells per frame but `sweD` covers eight times the area. Biasing the active LOD layer with μ should be done depending on the resolution n to avoid layers getting too small in the view port like `sweA` in figure 5.6.

One drawback that can be observed is that an overhead is introduced since each layer requires its own compute shader dispatch. Covering a total area of 819.2 m in `sweA` with 5 layers takes 0.1605 ms and in `sweF` it takes 0.1810 ms for just 1 layer. But in `sweF` more than double the amount of lattice cells (1 048 576) is processed in about the same time. The less work done by the GPU in a single dispatch, the less is the number of processed cells per time. To prevent getting low detail layers inefficient an attempt should be made to treat all layers in a single dispatch. Here, reading from and writing to different texture arrays for different layers poses a difficulty.

Besides the percentage deterioration, absolute times to cover the same area at a lower detail render the LOD system very flexible. The single highest

Name of the test	`sweA`	`sweB`	`sweC`	`sweD`	`sweE`	`sweF`	`sweG`
Set up parameters							
Shallow Water Size n	512	512	1024	1024	2048	2048	512
# Active LOD Layers M^{act}	5	5	5	5	5	5	1
Highest LOD Layer o_{act}	0	2	0	2	0	2	0
Properties of active layers 0 **to** 4							
[0] Spatial length l (m)	51.2	204.8	102.4	409.6	204.8	819.2	51.2
[0] Lattice size Δx (m)	0.1	0.4	0.1	0.4	0.1	0.4	0.1
[0] Sub step Size	512×512	256×256	1024×1024	512×512	2048×2048	1024×1024	512×512
[1] Spatial length l (m)	102.4	409.6	204.8	819.2	409.6	1638.4	–
[1] Lattice size Δx (m)	0.2	0.8	0.2	0.8	0.2	0.8	–
[1] Sub step Size	512×256	256×128	1024×512	512×256	2048×1024	1024×512	–
[2] Spatial length l (m)	204.8	819.2	409.6	1638.4	819.2	3276.8	–
[2] Lattice size Δx (m)	0.4	1.6	0.4	1.6	0.4	1.6	–
[2] Sub step Size	256×256	128×128	512×512	256×256	1024×1024	512×512	–
[3] Spatial length l (m)	409.6	1638.4	819.2	3276.8	1638.4	6553.6	–
[3] Lattice size Δx (m)	0.8	3.2	0.8	3.2	0.8	3.2	–
[3] Sub step Size	256×128	128×64	512×256	256×128	1024×512	512×256	–
[4] Spatial length l (m)	819.2	3276.8	1638.4	6553.6	3276.8	13107.2	–
[4] Lattice size Δx (m)	1.6	6.4	1.6	6.4	1.6	6.4	–
[4] Sub step Size	128×128	64×64	256×256	128×128	512×512	256×256	–
Stats							
Simulated cells per frame	507 904	126 976	2 031 616	507 904	8 126 464	2 031 616	262 144
Texture size (MB)	20.480	20.480	81.920	81.920	327.680	327.680	4.096
[0] Simulation Time (ms)	0.0586	0.0282	0.1795	0.0596	0.6574	0.1810	0.0573
[1] Simulation Time (ms)	0.0376	0.0200	0.0973	0.0379	0.3366	0.0996	–
[2] Simulation Time (ms)	0.0279	0.0172	0.0581	0.0300	0.1784	0.0586	–
[3] Simulation Time (ms)	0.0200	0.0159	0.0381	0.0197	0.0993	0.0402	–
[4] Simulation Time (ms)	0.0164	0.0161	0.0287	0.0161	0.0556	0.0284	–
Total Simulation Time (ms)	0.1605	0.0974	0.4017	0.1633	1.3273	0.4078	0.0573
`fftA` Combination Pass Time (ms)	0.8622	0.9285	0.9170	0.9439	1.0035	1.0214	0.6909
`fftE` Combination Pass Time (ms)	1.1436	1.1940	1.1955	1.2605	1.3202	1.3240	0.9375

Table 5.2: List of six different quality settings plus a reference setting with minimal shallow water layers along their time and memory metrics. Results for specific layers are written using an array like notation where [0] is the highest detail active LOD layer. Timings for the Output Combination Pass are given for the two FFT tests `fftA` and `fftE` (see table 5.1).

layer of `sweF` covers 204.8 m with a lattice size of 0.1 m and a simulation time of 0.6574 s. The same area is covered by three layers in `sweA` taking just 0.1241 s with the same lattice size and level of detail for an area of 51.2 m.

In theory the simulated domain can get very big as it grows quadratically with layers. However, at some point it becomes questionable whether it is useful as the lattice size grows at the same rate, and therefore, the level of detail gets very low.

To evaluate the impact of LOD layers on the *Output Combination Pass*, times were measured in combination with a low and high quality setting `fftA` and `fftE` of table 5.1. Two key aspects affect the time: The area of the screen that is covered by LOD layers, and the resolution and thereby size of the texture. The former is the reason why the time from `sweA` to `sweB` rises more than from `sweE` to `sweF`. In `sweE` the screen is covered nearly completely (see 5.6) and an increase of spatial coverage has a small impact.

Since `sweB` and `sweE` cover the same simulation domain of 3276.8 m, the time difference can be attributed to the second aspect of increased resolution. Besides the higher bandwidth usage that is expected for the much higher texture size, the `sweE` test has its *L2 Cache Hit Rate* decreased by approximately 25%. This indicates that sampling LOD layers for neighbouring pixels in screen space does not translate to closely neighboured sample locations in the simulation texture. Comparing `sweE` with `sweF` supports this assumption, because in `sweF` the *L2 Cache Hit Rate* is 13 percentage points higher than in `sweE`. Therefore, the LOD bias should be selected carefully. However, it would be best when the selection of the highest active LOD layer gets coupled with the resulting pixel size instead of using a formula based on altitude and a controllable LOD bias.

Overall the GPU implementation of the macroscopic LBM for the shallow water simulation yields promising and manageable timing for the usage in a video game. The added scalability of simulation time, memory usage and covered area introduced by the LOD system makes it even more attractive.

5.2.3 Adaptive Level of Detail

It was found that moving layers with the camera introduces an undesirable amount of complexity. Even in the final implementation, re-alignment created disturbances on the water surface. Instead of the alignment to the camera movement a different set up of stationary LOD layers should be employed. The exact nature should be adjusted to the usage scenario, but the core concept of doubling the spatial domain and time step and in turn halving the processed cells per frame stays the same.

This concept can be considered a basic framework towards spatial and temporal adaptivity built around the self-adjusting property of the macroscopic Lattice Boltzmann Method. It can be used for other refinement criteria instead of the distance to the camera. More detail could be used just in certain area of interests like near the shore. Even more sophisticated criteria like the closeness of a layer's content to the representable detail could be used to increase detail where the flow gets more complex.

Certainly, the proposed LOD system requires more attention regarding introduced errors if adopted for scientific purposes. Important considerations are the choice of boundary handling between layers or the copying of data between layers of different detail. A reason for surface disturbances in this work's implementation are mismatched sample locations of the seabed elevation. They are located at a cell's centre and after copying to another layer the sampled elevation is not identical any more. Precise interpolation or averaging of the simulated physical quantities should be introduced, to tackle this issue. This should be the researched in more detail in the future as it is out of the scope of this work.

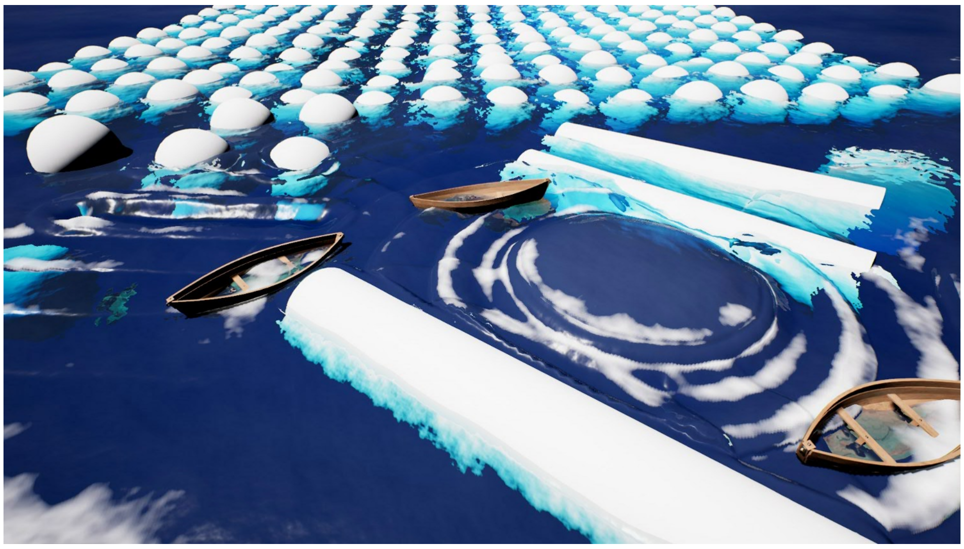


Figure 5.7: Two-way interaction of the simulated ocean surface and rigid bodies modelled by proxy objects.

5.3 Interaction with Rigid Bodies

With the physical variables depth and horizontal velocity in place they can be directly manipulated to cause an effect of rigid bodies colliding with the water surface. This is an advantage of the macroscopic formulation over to the standard LBM where this modification has to be integrated into the equilibrium distribution function [Oje13a]. In contrast to the pipe method, effects can be articulated in terms of a change of fluid velocity. This should be preferred over height modifications to keep the volume identical, unless adding or removing water from the system is desired.

The idea of physics proxy objects is to simplify the interaction with the ocean surface. Spheres offer the simplest form of collision detection. With the addition of capsules, benefits of sphere collision detection can be kept while it can be ensured to detect collisions even for large time steps. Applying procedural effects onto the surface within the collision volume allows to hide the simple nature of the proxy object. However, authoring effects on a pure mathematical basis is not very appealing from a designer's perspective. Instead, texture-based descriptions of effects could be introduced.

Figure 5.7 shows a scene with various proxy objects interacting with the shallow water simulation. Several sine wave effects introduce wave to the simulation by directly altering the height. Buoyancy is calculated and read back for a multitude of spheres, small boats and cylinders that reflect incoming waves simply by negating the incoming fluid's velocity. In figure 5.8 an in-editor view shows the placement of several proxy objects including spheres for the buoyancy calculation with a line representing the

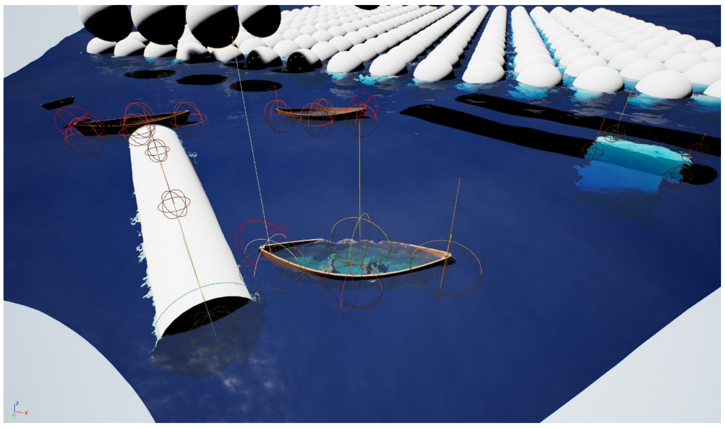


Figure 5.8: This screenshot of the UE editor shows the placement of proxy objects and force read back indicators.

magnitude of it. Besides buoyancy one could add other forces like drag or drift in the future.

For concave objects like the boat the problem arises that the water surface simply cuts through it. Solving this by actually displacing the water would require a very high simulation resolution that is not feasible. Therefore, the problem either has to be avoided by not using this kind of objects or the surface has to be removed during rendering, for example by using a depth mask made for this purpose.

Being able to directly map objects to compute shader groups of currently active sub steps is a form of coarse collision detection performed in advance. In the worst case, the same amount of lattice cells has to be updated as were during the simulation (see table 5.2). As it is the concept of the proxy objects to use a rather low amount of proxy objects and make up for it by employing procedural effects, the amount of objects processed by a single thread group naturally stays reasonable low. No test showed higher timings for the physics update as the shallow water simulation itself required. For the scene in figure 5.7 a maximum of ~0.25 ms was measured while using the most ambitious quality settings `sweE`. Reading back buoyancy of approximately 256 objects takes ~0.04 ms.

An issue arose since the shader compiler of the used Unreal Engine version requires to be Direct3D 11 compliant even if Direct3D 12 is used. During the physics update step, the depth and velocity of the shallow water texture has to be read, updated and written again. This is a complete local operation without the possibility for race conditions, but Direct3D 11 does not support reading and writing from 4 channel 32 bit textures [Mic18b]. Therefore, a second pass had to be used to write the physics results to both simulation texture arrays. This pass takes about 75% of the first physics update pass and adds an unnecessary overhead that needs to be avoided in the future. Ultimately, an attempt could be made to merge the physics update with the simulation step itself.

5.4 Projected Grid Approach

The biggest upsides of the projected grid approach are the simple implementation and the regular screen space grid that promises automatic adjustment of mesh detail level. However, moving the camera also moves the grid's vertices. Thus, the sampling locations of the height field change which results in the swimming artefact. Additionally, the camera position must be kept outside the maximum height offset by introducing a projector different from the camera. With the camera close to the surface, the resulting mesh then approaches a rectangle in world space no longer having the tight fit to screen space while still suffering from the swimming artefact.

With the polar meshing improvement the swimming artefact problem should be reduced during camera rotation. However, this can only work while the projector's position is identical to the camera's position. As soon as the projector differs, the origin of rotation is no longer the camera's position and the improvement fails to have any benefit. Additionally, the problem still occurs during camera movement. Overall, it adds a lot of complexity to the projected grid approach with very limited benefits.

Taking the minimum water height of a whole compute shader's group that covers multiple sample points as proposed in this work is a more universal improvement that covers all type of camera movements. But it only reduces the artefact's strength. It is still noticeable and not removed.

Even without camera movements, problems can arise. Figure 5.4b shows a particular problematic scene of very steep wave fronts that get very jaggy because the mesh resolution is too low.

The projected grid approach certainly has its downsides. Once improvements are introduced, the simple nature of the approach quickly gets lost. The more measures are taken to counter its deficiencies, it becomes questionable whether it is reasonable to stick to the projected grid approach or to employ a completely different method. This work's conclusion on the projected grid approach is that the decision to adopt it, should be based on the answer of the question, if one can live with its deficiencies or not.

6 Conclusion

This thesis examines the combination of deep water ocean waves and a shallow water simulation to render an ocean reacting to different environmental influences in real-time.

Deep water waves generated by a Fast Fourier Transform with inputs sampled from multiple wave spectra can depict various sea states. It allows to model the wind influence on the ocean and covers both local wind and distant storms that produce low frequency swells. Sampling spectra for seamless and immersive run-time animations requires special attention. A sampling strategy was proposed, suitable for the multi-band FFT approach which was adopted due its advantageous real-time properties. Additionally, the sampling was implemented on the GPU with performance results that highlight the potential of a full GPU solution. The greatest challenge and performance impact was found in the memory layout of the numerous FFT outputs. While some optimizations were suggested, future work could have a closer look on how to efficiently handle the large amount of outputs.

The shallow water simulation supplements the deep water animation with waves reacting to the terrain and rigid bodies. Fading deep water waves and transferring them to the fluid simulation allows the highly-controllable Fast Fourier Transform-based sea to react with the shore in a plausible way. However, in the present state this transition lacks controllability and needs to be improved in future work. For rigid bodies, a system of proxy objects using simple sphere-based collision detection and a preceding coarse GPU work distribution was developed. It is used to procedurally model effects of objects on the simulated surface and to read back a buoyancy force. Besides some technical problems, future work should improve the poor artistic control that is offered by the procedural effect description.

A recent advancement of the Lattice Boltzmann Method is employed to solve the shallow water equations. It is suitable for a GPU implementation, has clear stability conditions and a reduced memory requirement but is rather restrictive in terms of flow features. While a short comparison with other simulation methods was presented, a more in-depth evaluation is necessary to make a representative statement on which is the best choice.

Findings on the properties of the macroscopic LBM were used to describe a Level of Detail system that allows to cover a very large simulation domain while maintaining high details in camera vicinity. An evaluation of performance revealed possible improvements but confirmed its suitability for a parallel implementation. While the proposed LOD implementation turned out to be too complicated and difficult to maintain, the basic findings remain valid and could be the basis for elaborate LOD systems in the future. Employing it in a scientific context requires an analysis on accuracy, which would be an interesting research topic. Since the Lattice Boltzmann Method is still being actively researched, it is exciting to see what possibilities the future will bring.

While rendering of the ocean surface is done by the Unreal Engine, everything required to do so is covered by this work. Geometry of the ocean surface is created using a projected grid approach adjusted to the camera frustum. Here, minor improvements were achieved, but its deficiencies stay prominent, and the conclusion was made to either accept them or switch to a different geometry representation.

The total frame time of ocean simulation and rendering by Unreal Engine is approximately 2.37 ms for lower quality settings, 3.17 ms for great quality settings and 4.50 ms for some worst-case non-recommended settings. Times were measured on the device as specified in chapter 5. Although these times lie above the 2 ms frame budget for water rendering as specified in the requirements' analysis, the proposals drawn to improve performance give reason to believe that meeting the 2 ms frame budget is achievable. Certainly, tests need to be conducted on lower end hardware, but the existing options of scalability should be sufficient to simulate and render an ocean similarly albeit less detailed.

Adding more features that are missing due to the 2D nature of the simulation is worthwhile for future work. At the same time foam should be increased as it was indicated in this work. When reflecting on the insights gained during the research for this work one can hardly wait on new generations of hardware to throw off the shackles of two dimensions and enjoy the delights of three.

Glossary

Beaufort Scale The Beaufort Scale assigns 12 numbers to different wind scales and describes the appearance of the sea [Ste08, p. 44]. 37

Direct3D Direct3D is a graphics application programming interface from *Microsoft* to use the rendering pipeline or compute shaders [Mic20]. 35, 80

Fast Fourier Transform The Fast Fourier Transform is an algorithm to efficiently compute the discrete Fourier Transform. In the context of this work it is connected with the approach of Tessendorf [Tes01] to synthesize a tileable patch of ocean surface. 1, 13, 19, 42, 67, 68, 82

Froude number The Froude number is a non-dimensional ratio of the speed of waves to their maximum speed [Bee97, p. 174]. 28

Gerstner Wave A Gernster Wave is a parametric description of a surface wave with a trochoidal shape [Bee97, p. 344]. 4, 5, 13, 72

Octave GNU Octave is a free software and an interpreted language to perform numerical computations. 42

Index of abbreviations

API application programming interface 35

BRDF bidirectional reflectance distribution function 6, 33, 63

CPU central processing unit 35, 42, 53, 56, 58, 68

FFT Fast Fourier Transform 2, 5, 8, 9, 14, 18–21, 33, 35, 38, 40–45, 54, 55, 63, 67, 69–74, 77, 82

GDC Game Developers Conference 5, 6

GPU graphics processing unit 2, 4, 7–9, 35, 37, 41–43, 47, 52, 56–58, 68, 69, 71, 75, 76, 78, 82

HLSL High Level Shading Language 35, 42, 63, 64

JONSWAP Joint North Sea Wave Project 15–17, 20, 37–39, 42

LBM Lattice Boltzmann method 2, 8, 9, 24, 25, 27, 28, 30, 35, 44, 46, 49, 55, 73, 75, 78, 79, 82

LOD level of detail 2, 35, 47–52, 55–57, 59, 61, 62, 73, 75–78, 83, E

SIGGRAPH Special Interest Group on Graphics and Interactive Techniques 33, 34

SWE shallow water equations 7–9, 26

UE Unreal Engine 35, 67, 69, 80

List of Symbols

c	Celerity, phase speed, wave speed
d	Depth
f	Frequency
g	Gravity $g \approx 9.81$
h	Height
ω	Angular frequency
ω_p	Angular peak frequency
T	Period
p	Pressure
$\boldsymbol{c}$	Point on the projected grid
ρ	Density
t	Time
$\boldsymbol{u}$	Velocity
λ	Wavelength
k	Wave number
$\boldsymbol{k}$	Wave vector
$\boldsymbol{x}$	Position in $\mathbb{R}^2$
α	Equilibrium constant controlling spectral intensity
a	Wave amplitude
β	Exponential growth factor
Dir	Directional spreading function
$\boldsymbol{D}$	Horizontal displacement
λ_d	Displacement scaling factor
f_{L}	Lower bound frequency
f_{fit}	Lower bound fitting factor
f^{max}	Maximum frequency of a band
f^{min}	Minimum frequency of a band
f_p	Peak frequency
f_p^{st}	Static peak frequency
f_{U}	Upper bound frequency
Γ	Gamma function
$\tilde{h}$	Fourier amplitude

H_s	Significant wave height
$H_s{}^{\text{st}}$	Static significant wave height
$\boldsymbol{J}$	Jacobian matrix of ocean displacement
J	Determinant of jacobian matrix
L	Spatial length
Δl	Mesh spacing
λ_k	Grid to wave vector scaling factor
m	Index of a band
M	Amount of bands in multi-band approach
N	Fourier grid size
$\omega^{\max}$	Maximum angular frequency of a band
ω_p^{st}	Static angular peak frequency
γ	Peak enhancement factor
p_{L}	Lower bound peak frequency factor
p_{U}	Upper bound peak frequency factor
γ_{s}	Peak sampling enhancement factor
σ	Width of spectral peak
T_p	Peak period
T_p^{st}	Static peak period
S	Spectral density function
s	Spread
θ	Angle between wave and wind direction
$k^{\max}$	Maximum wave number of a band
$\boldsymbol{u}_w$	Wind direction
F	Wind fetch
F^{st}	Static wind fetch
U	Wind speed
U^{st}	Static wind speed
C_b	Bed friction coefficient
C_i	Lattice direction force constant
d_d	Desired depth that can be simulated
$\overline{d}$	Semi implicit depth average
e	Particle speed
$\boldsymbol{e}_i$	i-th particle velocity vector
f_i	i-th particle distribution
f_i^{eq}	Local equilibrium distribution function
λ_{swf}	Shallow water foam intensity
$\boldsymbol{F}$	Force term
Fr	Froude number
l	Shallow water grid spatial length
λ_{dw}	Global deep water influence on shallow water
λ_i	Lattice direction equilibrium constant
λ_{zone}	Deep water transfer zone factor

h_{cam}	Altitude of the camera
μ	LOD bias
o	Index of a LOD layer
M^{act}	Number of active LOD layers
M^{sub}	Number of substeps
Δt_f	Frame time
n	Shallow water grid size
ν	Eddy or kinematic viscosity
$\mathit{\Omega}$	Collision operator
Re	Reynolds number
τ	Single relaxation time
τ_b	Bed shear stress
$\mathit{\Delta} t$	Delta time
V	Volume
$\mathit{\Delta} x$	Lattice size
z_b	Bed elevation
z^{p}	Sea bed peak transfer zone
z^{w}	Sea bed transfer zone width

Bibliography

[Ang+18] Nigel Ang et al. “The Technical Art of Sea of Thieves”. In: *ACM SIGGRAPH 2018 Talks on - SIGGRAPH ’18*. Vancouver, British Columbia, Canada: ACM Press, 2018, pp. 1–2.

[Bav19] Louis Bavoil. *The Peak-Performance-Percentage Analysis Method for Optimizing Any GPU Workload*. June 2019. URL: `https://developer.nvidia.com/blog/the-peak-performance-analysis-method-for-optimizing-any-gpu-workload/` (visited on 09/16/2020).

[Bee97] Tom Beer. *Environmental Oceanography*. 2nd ed. CRC Series in Marine Science. Boca Raton: CRC Press, 1997.

[BL17] Jean-Normand Bucci and Nicolas Longchamps. *From Shore to Horizon: Creating a Practical Tessellation Based Solution*. 2017. URL: `https://www.gdcvault.com/play/1024591/From-Shore-to-Horizon-Creating` (visited on 09/02/2020).

[BNH10] Eric Bruneton, Fabrice Neyret, and Nicolas Holzschuch. “Real-Time Realistic Ocean Lighting Using Seamless Transitions from Geometry to BRDF”. In: *Computer Graphics Forum* 29.2 (May 2010), pp. 487–496.

[Bow13] Huw Bowles. *Oceans on a Shoestring: Shape Representation, Meshing and Shading*. Anaheim, California, July 2013. URL: `http://advances.realtimerendering.com/s2013/` (visited on 09/09/2020).

[Bri08] Robert Bridson. *Fluid Simulation for Computer Graphics*. Wellesley, Mass: A K Peters, 2008.

[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*. SCA ’10. Goslar, DEU: Eurographics Association, 2010, pp. 197–206.

[CMK15] Nuttapong Chentanez, Matthias Muller, and Tae-Yong Kim. "Coupling 3D Eulerian, Heightfield and Particle Methods for Interactive Simulation of Large Scale Liquid Phenomena". In: *IEEE Transactions on Visualization and Computer Graphics* 21.10 (Oct. 2015), pp. 1116–1128.

[CS09] H Cords and O Staadt. "Real-Time Open Water Environments with Interacting Objects". In: (2009), p. 8.

[Dar+11] E. Darles et al. "A Survey of Ocean Simulation and Rendering Techniques in Computer Graphics". In: *Computer Graphics Forum* 30.1 (Mar. 2011), pp. 43–60.

[Epi] Epic Games Inc. *Parallel Rendering Overview | Unreal Engine Documentation*. URL: `https://docs.unrealengine.com/en-US/Programming/Rendering/ParallelRendering/index.html` (visited on 09/10/2020).

[FR86] Alain Fournier and William T. Reeves. "A Simple Model of Ocean Waves". In: *SIGGRAPH Comput. Graph.* 20.4 (Aug. 1986), pp. 75–84.

[Fré06] Jocelyn Fréchet. "Realistic Simulation of Ocean Surface Using Wave Spectra". In: *Proceedings of the First International Conference on Computer Graphics Theory and Applications*. Setúbal, Portugal: SciTePress - Science and and Technology Publications, 2006, pp. 76–83.

[Gam18] Thomas Gamper. "Ocean Surface Generation and Rendering". PhD thesis. Favoritenstrasse 9-11/E193-02, A-1040 Vienna, Austria: Institute of Computer Graphics and Algorithms, Vienna University of Technology, Sept. 2018.

[Gov+08] Naga K. Govindaraju et al. "High Performance Discrete Fourier Transforms on Graphics Processors". In: *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*. Austin, TX: IEEE, Nov. 2008, pp. 1–12.

[HBB03] Anders Hast, T Barrera, and E Bengtsson. "Shading by Spherical Linear Interpolation Using de Moivre's Formula". In: *WSCG'03, Short Paper*. University of Gävle, Ämnesavdelningen för datavetenskap, 2003, pp. 57–60.

[HO73] K. Hasselmann and Dirk Olbers. "Measurements of Wind-Wave Growth and Swell Decay during the Joint North Sea Wave Project (JONSWAP)". In: *Ergänzung zur Deut. Hydrogr. Z., Reihe A (8)* 12 (1973), pp. 1–95.

[Joh04] C. Johanson. "Real-Time Water Rendering: Introducing the Projected Grid Concept". PhD thesis. Univ., 2004.

[Kel17] Timo Kellomäki. "Fast Water Simulation Methods for Games". In: *Computers in Entertainment* 16.1 (Dec. 2017), pp. 1–14.

[Kry16] Yury A. Kryachko. "Sea Surface Visualization in World of Warships". In: *ACM SIGGRAPH 2016 Talks on - SIGGRAPH '16*. Anaheim, California: ACM Press, 2016, pp. 1–2.

[Lan64] C. Lanczos. "A Precision Approximation of the Gamma Function". In: *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis* 1.1 (1964), pp. 86–96.

[LCS63] M. S. Longuet-Higgins, D. E. Cartwright, and N. D. Smith. "Observations of the Directional Spectrum of Sea Waves Using the Motions of a Floating Buoy". In: *Ocean Wave Spectra*. New Jersey, USA: Prentice-Hall, Inc., Jan. 1963, pp. 111–136.

[LeB+12] Graham LeBlanc et al. "Multi-Band Fourier Synthesis of Ocean Waves". In: *Journal of Graphics Tools* 16.2 (June 2012), pp. 57–70.

[Man+17] P.-L. Manteaux et al. "Adaptive Physically Based Models in Computer Graphics". In: *Computer Graphics Forum* 36.6 (Sept. 2017), pp. 312–337.

[MDH07] Xing Mei, Philippe Decaudin, and Bao-Gang Hu. "Fast Hydraulic Erosion Simulation and Visualization on GPU". In: *15th Pacific Conference on Computer Graphics and Applications (PG'07)*. Maui, HI, USA: IEEE, Oct. 2007, pp. 47–56.

[MFC06] Marcelo M. Maes, Tadahiro Fujimoto, and Norishige Chiba. "Efficient Animation of Water Flow on Irregular Terrains". In: *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia - GRAPHITE '06*. Kuala Lumpur, Malaysia: ACM Press, 2006, p. 107.

[Mic18a] Microsoft Corporation. *Read Back Data via a Buffer*. Dec. 2018. URL: `https://docs.microsoft.com/en-us/windows/win32/direct3d12/readback-data-using-heaps` (visited on 09/27/2020).

[Mic18b] Microsoft Corporation. *Typed Unordered Access View (UAV) Loads*. May 2018. URL: `https://docs.microsoft.com/en-us/windows/win32/direct3d12/typed-unordered-access-view-loads` (visited on 09/27/2020).

[Mic20] Microsoft Corporation. *Direct3D*. July 2020. URL: `https://docs.microsoft.com/en-us/windows/win32/direct3d` (visited on 09/27/2020).

[Moh19] A. A. Mohamad. *Lattice Boltzmann Method: Fundamentals and Engineering Applications with Computer Codes*. London: Springer London, 2019.

[MT19] Mark Mihelich and Tim Tcheblokov. *Advanced Graphics Techniques Tutorial: Wakes, Explosions and Lighting: Interactive Water Simulation in 'Atlas'*. San Francisco, Mar. 2019. URL: `https://gdcvault.com/play/1025819/Advanced-Graphics-Techniques-Tutorial-Wakes` (visited on 09/03/2020).

[Mun20] Robert Munafo. *Coefficients for the Lanczos Approximation to the Gamma Function*. Mar. 2020. URL: `https://mrob.com/pub/ries/lanczos-gamma.html%5C#code` (visited on 09/12/2020).

[OH95] J.F. O'Brien and J.K. Hodgins. "Dynamic Simulation of Splashing Fluids". In: *Proceedings Computer Animation'95*. Geneva, Switzerland: IEEE Comput. Soc. Press, 1995, pp. 198–205.

[Oje13a] Ojeda. "Hybrid Particle Lattice Boltzmann Shallow Water for Interactive Fluid Simulations:" in: *Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications*. Barcelona, Spain: SciTePress - Science and and Technology Publications, 2013, pp. 217–226.

[Oje13b] Jesús Ojeda. "Efficient Algorithms for the Realistic Simulation of Fluids". PhD thesis. Universitat Politècnica de Catalunya, 2013.

[Ola+13] Michel Olagnon et al. "West Africa Swell Spectral Shapes". In: *Volume 2B: Structures, Safety and Reliability*. Nantes, France: American Society of Mechanical Engineers, June 2013, V02BT02A029.

[PJN55] Willard J. Pierson, Richard W. James, and Gerhard Neumann. *Practical Methods for Observing and Forecasting Ocean Waves by Means of Wave Spectra and Statistics*. Vol. Publication 603. (Hydrographic Office). Washington D.C.: U.S. Government Printing Office, 1955.

[PM63] Willard J Pierson and Lionel Moskowitz. "A Proposed Spectral Form for Fully Developed Wind Seas Based on the Similarity Theory of S. A. Kitaigorodskii". In: (1963), p. 32.

[PP20] Sekil Park and Jinah Park. "Realistic Simulation of Mixed Sea Using Multiple Spectrum-Based Wave Systems". In: *SIMULATION* 96.3 (Mar. 2020), pp. 281–296.

[Qui13] Inigo Quilez. *Smooth Minimum*. 2013. URL: `https://www.iquilezles.org/www/articles/smin/smin.htm` (visited on 09/16/2020).

[RW04] Lennart Rade and Bertil Westergren. *Mathematics Handbook for Science and Engineering*. Fifth. Springer-Verlag Berlin Heidelberg, 2004.

[Ste08] Robert H. Stewart. *Introduction to Physical Oceanography*. Available electronically from http://hdl.handle.net/1969.1/160216, Jan. 2008.

[Tch15] Tim Tcheblokov. *Ocean Simulation and Rendering in War Thunder*. 2015. URL: `http://developer.download.nvidia.com/assets/gameworks/downloads/regular/events/cgdc15/CGDC2015_ocean_simulation_en.pdf` (visited on 09/03/2020).

[Tes01] Jerry Tessendorf. "Simulating Ocean Water". In: *SIG-GRAPH'99 Course Note* (Jan. 2001).

[Thu+07] Nils Thurey et al. "Real-Time Breaking Waves for Shallow Water Simulations". In: *15th Pacific Conference on Computer Graphics and Applications (PG'07)*. Maui, HI, USA: IEEE, Oct. 2007, pp. 39–46.

[Totnd] Viktor T. Toth. *The Gamma Function*. n.d. URL: `http://www.rskey.org/CMS/index.php/the-library/11`.

[Tub10] Kevin Tubbs. "Lattice Boltzmann Modeling for Shallow Water Equations Using High Performance Computing". PhD thesis. LSU Doctoral Dissertations: Louisiana State University, Agricultural, and Mechanical College, 2010.

[Weind] Eric W. Weisstein. *Point-Line Distance–3-Dimensional*. n.d. URL: `https://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html` (visited on 09/17/2020).

[Zho04] Jian Guo Zhou. *Lattice Boltzmann Methods for Shallow Water Flows*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[Zho11] Jian Guo Zhou. "Enhancement of the LABSWE for Shallow Water Flows". In: *Journal of Computational Physics* 230.2 (Jan. 2011), pp. 394–401.

[Zho19a] Jian Guo Zhou. "Macroscopic Lattice Boltzmann Method (MacLAB)". In: *arXiv:1901.02716 [physics]* (Jan. 2019).

[Zho19b] Jian Guo Zhou. "Macroscopic Lattice Boltzmann Method for Shallow Water Equations (MacLABSWE)". In: *arXiv:1902.02999 [physics]* (Feb. 2019).

[ZL13] Jian Guo Zhou and Haifei Liu. "Determination of Bed Elevation in the Enhanced Lattice Boltzmann Method for the Shallow-Water Equations". In: *Physical Review E* 88.2 (Aug. 2013).