Water Surface Wavelets

STEFAN JESCHKE, NVIDIA
TOMÁŠ SKŘIVAN, IST Austria
MATTHIAS MÜLLER-FISCHER, NVIDIA
NUTTAPONG CHENTANEZ, NVIDIA
MILES MACKLIN, NVIDIA
CHRIS WOJTAN, IST Austria

Fig. 1. A frame from a real-time wave animation with the simulation grid overlaid on top. We compute the wave motions at a resolution several orders of magnitude finer than the simulation variables.

The current state of the art in real-time two-dimensional water wave simulation requires developers to choose between efficient Fourier-based methods, which lack interactions with moving obstacles, and finite-difference or finite element methods, which handle environmental interactions but are significantly more expensive. This paper attempts to bridge this long-standing gap between complexity and performance, by proposing a new wave simulation method that can faithfully simulate wave interactions with moving obstacles in real time while simultaneously preserving minute details and accommodating very large simulation domains.

Previous methods for simulating 2D water waves directly compute the change in height of the water surface, a strategy which imposes limitations based on the CFL condition (fast moving waves require small time steps) and Nyquist's limit (small wave details require closely-spaced simulation variables). This paper proposes a novel wavelet transformation that discretizes the liquid motion in terms of amplitude-like functions that vary over *space, frequency, and direction*, effectively generalizing Fourier-based methods to handle local interactions. Because these new variables change much more slowly over space than the original water height function, our change of variables drastically reduces the limitations of the CFL condition and Nyquist limit, allowing us to simulate highly detailed water waves at very large visual resolutions. Our discretization is amenable to fast summation and easy to parallelize. We also present basic extensions like pre-computed wave paths and two-way solid fluid coupling. Finally, we argue that our discretization provides a convenient set of variables for artistic manipulation, which we illustrate with a novel wave-painting interface.

Authors' addresses: Stefan Jeschke, NVIDIA, jeschke@stefan-jeschke.com; Tomáš Skřivan, IST Austria, tomas.skrivan@ist.ac.at; Matthias Müller-Fischer, NVIDIA, matthiasm@nvidia.com; Nuttapong Chentanez, NVIDIA, nchentanez@nvidia.com; Miles Macklin, NVIDIA, mmacklin@nvidia.com; Chris Wojtan, IST Austria, wojtan@ist.ac.at.

0730-0301/2018/8-ART94
https://doi.org/10.1145/3197517.3201336

CCS Concepts: • **Computing methodologies → Physical simulation**;

Additional Key Words and Phrases: Water animation, real-time animation, natural phenomena

ACM Reference Format:
Stefan Jeschke, Tomáš Skřivan, Matthias Müller-Fischer, Nuttapong Chentanez, Miles Macklin, and Chris Wojtan. 2018. Water Surface Wavelets. *ACM Trans. Graph.* 37, 4, Article 94 (August 2018), 13 pages. https://doi.org/10.1145/3197517.3201336

1 INTRODUCTION

This paper concerns the efficient and physically plausible animation and art-direction of water surface waves at large scales. Current solutions to this problem invoke numerical solutions to partial differential equations (like the shallow water equations or dispersive wave equations), or analytical solutions based on Fourier transforms. Numerical solutions excel at handling water interactions with moving obstacles, but they become expensive to compute when scaling to very large simulation domains with small (high frequency) wave details. Conversely, Fourier summation techniques excel at simulating very large domains with high-frequency details, but they cannot easily incorporate complex environmental interactions like moving boundaries and spatially-varying wind.

Our work proposes a novel transformation to speed up the computation of water surface waves. Instead of discretizing the wave height and momentum at each point on a grid (like previous finite-difference methods), or discretizing wave amplitudes as a function of frequency and direction (like previous Fourier-based methods), we introduce a wavelet transformation that discretizes the wave amplitudes as a function of space, frequency, and direction *combined*. The variables resulting from this discretization change much more slowly over space than the original water wave height function, so we can represent the same amount of information with fewer variables. The new lower-frequency simulation is also less sensitive to traditional frequency-based limitations like the CFL condition and the Nyquist limit, which convert the maximum spatial frequency into limitations on time step size and visual detail. As a consequence, our discretization permits both high-resolution wave details (like Fourier-based methods) as well as local wave interactions with moving obstacles.

We derive new equations for propagating these local frequency dependent amplitudes through space; these equations result in simple 2D advection and diffusion operations that can be parallelized easily on graphics hardware, giving us interactive frame rates. We also present basic extensions to our simulator, like pre-computed wave paths and two-way solid fluid coupling. Finally, we found that this new representation provides a convenient artistic interface for hand-tuning the motion of complicated ocean simulations, and we show a prototype wave-painting interface for initializing simulations or overriding the physics with scripted motions.

The contributions of our paper are:

- **Eulerian Wavelet Transformation:** A new theoretical model for water wave transport based on the theory of slowly modulated waves.
- **Low-frequency simulation variables:** Our discretization relies on functions that vary more slowly over space than the water height itself, so we can represent them on lower resolution grids. This change of variables allows more efficient computation and larger computational domains (Figure 1).
- **Novel artistic control:** In addition to determining the amplitude function using the physical equations of motion, we also experiment with overwriting these wave amplitudes for artistic effect. We show how our method can be used to pre-compute wave scenes faster and more easily than previous work, and we present an interactive painting interface for designing spatially-varying ocean waves.

2 RELATED WORK

Since the early days of computer animation [Schachter 1980], the main strategy for recovering the shape and motion of surface water geometry has been to approximately solve the Navier-Stokes equations. There are numerous ways to approximate these equations, and this discussion divides the techniques into analytical "spectrum-based" approaches, direct numerical simulation of partial differential equations, and hybrid approaches. We close this section by discussing methods for art-directing wave simulations.

2.1 Spectrum-based approaches

Works in both physics and computer graphics employ numerous theoretical assumptions to the Navier-Stokes equations in order to reduce the complexity and make them analytically tractable. Some common assumptions in computer graphics are deep water, potential flow, small amplitudes, and periodic boundary conditions. These approaches take advantage of analytical solutions to the Navier-Stokes equations on a 2D height field in order to express the motion of the ocean in the form of sines and cosines, while sacrificing the ability to simulate arbitrary fluid motion [Hinsinger et al. 2002; Horvath 2015; Mastin et al. 1987; Tessendorf 2004b]. Jeschke & Wojtan [2015] simulate the propagation of wavefronts throughout a static environment, in order to extend spectrum-based approaches to handle complex boundaries. However, the method is limited to pre-computation, and does not allow interactions with moving obstacles.

We call this technique of discretizing a Fourier-transform the "spectrum-based" approach. These methods exhibit theoretically unlimited visual detail, in the sense that they can animate arbitrarily high frequency waves without impacting the method's accuracy or stability. Similarly, fast wave speeds are trivial to simulate, because the motion is independent of the time step size. However, because the derivation of these methods makes several assumptions about the underlying flow, they tend to have limitations like the inability to realistically interact with complex boundaries.

2.2 Numerical solutions to Partial Differential Equations

A great way to get around the limitations of spectrum-based methods is to directly simulate a two-dimensional version of the Navier-Stokes equations using numerical algorithms. Some approaches discretize simplified wave equations [Kass and Miller 1990; Thuerey et al. 2010; Yang et al. 2016] or thin plate equations [Yu et al. 2012]; these simplified equations are easier to implement, but the resulting behavior is fundamentally different from actual water waves. The

work of Tessendorf [2004a; 2014], which discretizes a linearized Bernoulli equation, exhibits more realistic wave dispersion but still falls short of physically correct motion (as discussed in [Canabal et al. 2016]). Some researchers have also introduced direct numerical simulators based on the lattice Boltzmann method (LBM) [Geist et al. 2010], which requires a careful tuning of the LBM collision matrix to yield realistic wave speeds. Convolution-based approaches [Loviscach 2002; Ottosson 2011] aim to achieve the correct dispersion relation, but they must cope with the practical difficulties of large kernels occupying the entire simulation domain. Recently, Canabal et al. [2016] overcame these difficulties with a combination of pyramid filters and shadowed convolution operations.

Each of these methods tend to be much more flexible than spectrum-based methods, because they make minimal assumptions about the environment. For example, spectrum-based methods have difficulties with simulating obstacle interactions because their derivation assumes periodic boundaries, while direct simulation approaches have no such limitation. On the other hand, these methods must simulate wave propagation by iterating local kernel operations instead of simply plugging a time parameter into a cosine function. All of these methods discretize wave heights and momentum directly on an Eulerian grid or mesh, and, as a consequence, the resolution of the grid is directly tied to the amount of visible detail in the waves. Nyquist's theorem requires that the grid must be fine enough to resolve the highest frequency in the heightfield; otherwise aliasing and instability may occur. Similarly, the stability of an explicitly integrated wave simulation is also intimately related to the grid spacing by the CFL condition. This stability problem can be overcome by implicit integration, at the expense of additional computation and a more complex and less GPU-friendly implementation.

Fully three-dimensional techniques for liquid simulation are outside the scope of this work. We recommend that interested readers consult the text by Bridson [2015].

2.3 Hybrid approaches

Our method combines the flexibility of numerical approaches with the stability and visual detail of spectrum-based approaches, but we are not the first to do so. Yuksel et al [2007] proposes wave particles which represent a local wave crest and move with a predetermined wave speed c. We can view this approach as a local spectrum-based method if we use many different particle sizes and set c equal to the analytical phase speed. Taking this further, Jeschke & Wojtan [2017] introduced wave packets which propagate at the theoretical group speed and contain a small train of waves traveling at the theoretical phase speed. These approaches inherit some advantages of spectrum-based methods, like the numerical stability and theoretically accurate wave speeds. At the same time, they avoid the complications with spectrum-based waves by breaking the global cosine waves into a train of shorter wave components that are free to interact with obstacles. Our method provides similar advantages, but it is Eulerian rather than Lagrangian (its degrees of freedom are associated with regions of space rather than the wave motion itself). Consequently, our method maps to GPU hardware more easily, the computational complexity is constant as it does not vary with a number of particles, and it trivially interfaces with texture maps for easy artistic control.

2.4 Art-directing waves

Previous works have investigated liquid control by directly editing keyframes [McNamara et al. 2004; Shi and Yu 2005], by sculpting fluid interactively [Manteaux et al. 2016; Pan et al. 2013], by introducing guide forces on particles [Thuerey et al. 2009] or meshes [Raveendran et al. 2012], by composing together convenient flow primitives [Chenney 2004], or by interpolating between simulations and existing motions [Raveendran et al. 2014; Thuerey 2016]. Some works enhance existing animations with additional waves on surfaces [Angst et al. 2008; Kim et al. 2013] or near boundaries [Jeschke and Wojtan 2015, 2017]. Horvath et al. [Horvath 2015] explores the control of ocean spectra in great detail, and Nielsen et al. [Nielsen et al. 2013] investigates how to direct a spectrum-based method based on a scripted height field. We are unaware of any prior methods that locally direct the amplitudes of directional water waves, as we propose.

3 THEORY

We first explain the motivation for our new wavelet-based water wave discretization in Section 3.1. Then, we derive a partial differential equation describing the evolution of these water wavelets in Section 3.2. Finally, we discuss the validity of the derivation and provide some interpretations for the method in Section 3.3.

3.1 Motivation

Current numerical methods for water wave simulation rely either on discretizations of partial differential equations (PDEs), or on spectrum-based methods. Methods based on discretized PDEs approximate some differential equation that describes how the wave height $\eta(\boldsymbol{x}, t)$ evolves over time:

$$\frac{\partial \eta(\boldsymbol{x}, t)}{\partial t} = \dots \tag{1}$$

where the right hand side is defined by the particular wave model (shallow water equations, Bernoulli equation, etc.), and environmental interactions are encoded in the PDE's boundary conditions. These discretizations sample $\eta(\boldsymbol{x}, t)$ over space with a grid spacing equal to Δx. In order to avoid aliasing and faithfully reproduce high-frequency details, The Nyquist-Shannon sampling theorem [Shannon 1949] requires that Δx is less than half of the shortest wavelength in $\eta(\boldsymbol{x}, t)$. If $\eta(\boldsymbol{x}, t)$ contains interesting high-frequency details, then the samples of $\eta(\boldsymbol{x}, t)$ must be very close together, and thus Δx must be very small. Practically, decreasing Δx requires either more samples or a smaller simulation domain, so the Nyquist theorem effectively limits the visual detail that a simulator can produce. Highly detailed visuals can be obtained by decreasing Δx at the expense of vastly increased computation time and memory.

Spectrum-based methods for animating water waves [Tessendorf 2004b] remove these problems by avoiding a spatial discretization altogether. They rely on linear wave theory [Johnson 1997], which describes the wave height dynamics in terms of frequencies instead

of partial derivatives:

$$\eta_c(\boldsymbol{x}, t) = \int_{\mathbb{R}^2} A(\boldsymbol{k}) e^{i(\boldsymbol{k}\cdot\boldsymbol{x} - \omega(k)t)}\, d\boldsymbol{k}. \tag{2}$$

Here, $\eta_c(\boldsymbol{x}, t)$ is a complex function that varies over two-dimensional space and time, and we can get the wave height by taking its real part, $\eta(\boldsymbol{x}, t) = \mathrm{Re}\ \eta_c(\boldsymbol{x}, t)$. The wavevector $\boldsymbol{k}$ is a two-dimensional frequency function, the wavenumber $k = |\boldsymbol{k}|$ represents a scalar frequency, and $\hat{\boldsymbol{k}} = \boldsymbol{k}/k$ is the wave direction. The exponential term in this equation represents a traveling wave, and $A(\boldsymbol{k})$ represents its amplitude. The angular frequency

$$\omega(k) = \sqrt{gk + \sigma k^3} \tag{3}$$

encodes the speed of each wave based on its wavenumber k, gravity g, and surface tension σ. Spectrum-based methods compute the wave height by discretizing the integral over all of these two-dimensional waves, instead of discretizing a differential equation. This solution works perfectly for ideal situations with periodic domains, and no boundaries or interacting obstacles. However, these spectrum-based methods become impractical or impossible in less constrained scenarios.

To summarize, PDE-based methods for animating water waves excel in the simulation of low-frequency wavefunctions with complicated environmental interactions, while spectrum-based methods are ideal for simulating highly detailed wavefunctions undergoing simple motion without any boundary interactions. In the following section, we derive a hybrid discretization that relies on discretized PDEs to simulate low frequency motion and uses spectrum-based techniques for simulating the high frequencies. This strategy allows us to simulate wavelengths far shorter than Δx interacting with complex environments, while eliminating the previously mentioned problems associated with simulating highly detailed waves.

Our derivation relies on the Gabor wavelet transform, which effectively transforms Equation 2 (with amplitudes A depending only on $\boldsymbol{k}$ and independent of $\boldsymbol{x}$ and t) to a similar one with amplitudes $\mathcal{A}$ depending on $\boldsymbol{k}$, $\boldsymbol{x}$, and t:

$$\eta_c(\boldsymbol{x}, t) = \int_{\mathbb{R}^2} \mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) e^{i(\boldsymbol{k}\cdot\boldsymbol{x} - \omega(k)t)}\, d\boldsymbol{k}. \tag{4}$$

Not only does this new spatially-varying amplitude $\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t)$ allow more *local* control over the simulation compared to $A(\boldsymbol{k})$, but we prove that $\mathcal{A}$ is guaranteed to have lower frequency content than the wave height function $\eta(\boldsymbol{x}, t)$. We will take advantage of this fact to forestall problems related to the Nyquist limit and create highly efficient and detailed wave simulations.

3.2 Derivation

The Gabor transform [Gabor 1946] of the water height η_c is

$$\zeta(\boldsymbol{x}, \boldsymbol{k}, t) = \frac{1}{(2\pi)^2} \int_{\mathbb{R}^2} \eta_c(\boldsymbol{y}, t) e^{-\frac{|\boldsymbol{y}-\boldsymbol{x}|^2}{2s^2}} e^{-i\boldsymbol{k}\cdot\boldsymbol{y}}\, d\boldsymbol{y} \tag{5}$$

where the first exponential term in the integrand is a Gaussian centered at position $\boldsymbol{x}$ with standard deviation s, and the second exponential term is a static wave with wavevector $\boldsymbol{k}$. We can think of Equation 5 as an inner product between η_c and a Gaussian wavelet, so the number $\zeta(\boldsymbol{x}, \boldsymbol{k}, t)$ tells how much the water height η_c behaves like a wave with wave-vector $\boldsymbol{k}$ in the vicinity of point $\boldsymbol{x}$.

We can also invert the Gabor transform:

$$\eta(\boldsymbol{x}, t) = \mathrm{Re} \int_{\mathbb{R}^2} \zeta(\boldsymbol{x}, \boldsymbol{k}, t) e^{i\boldsymbol{k}\cdot\boldsymbol{x}}\, d\boldsymbol{k}, \tag{6}$$

which reminds us of a Fourier transform with ζ acting like an amplitude that now depends on $\boldsymbol{x}$ and t as well as $\boldsymbol{k}$. We introduce the change of variables $\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) = \zeta(\boldsymbol{x}, \boldsymbol{k}, t) e^{i\omega(k)t}$ to obtain an analogue to the dynamic wave evolution in Equation 2:

$$\boxed{\eta(\boldsymbol{x}, t) = \mathrm{Re} \int_{\mathbb{R}^2} \mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) e^{i(\boldsymbol{k}\cdot\boldsymbol{x} - \omega(k)t)}\, d\boldsymbol{k},} \tag{7}$$

now with $\mathcal{A}$ playing the role of an amplitude that varies over *space* and *time* as well as wavevector. By plugging Equation 2 into Equation 5, we obtain an equation for the time evolution of $\mathcal{A}$:

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) = \int_{\mathbb{R}^2} \tilde{A}(\boldsymbol{l}, \boldsymbol{k}) e^{i((\boldsymbol{l}-\boldsymbol{k})\cdot\boldsymbol{x} - (\omega(l) - \omega(k))t)}\, d\boldsymbol{l}, \tag{8}$$

$$\tilde{A}(\boldsymbol{l}, \boldsymbol{k}) = A(\boldsymbol{l}) s^2 e^{-\frac{1}{2}s^2(\boldsymbol{l}-\boldsymbol{k})^2}. \tag{9}$$

Although this equation completely prescribes the evolution of $\mathcal{A}$, its integral form makes it difficult to impose boundary conditions and generally inconvenient to discretize. We prefer to work with a differential equation instead, which we derive by taking the time derivative of Equation 8 and expressing it in terms of its spatial derivative. Linearizing the dispersion relation $\omega(l) \approx \omega(k) + \omega'(k)\hat{\boldsymbol{k}} \cdot (\boldsymbol{l} - \boldsymbol{k})$ gives us a first order partial differential equation for the evolution of $\mathcal{A}$:

$$\boxed{\frac{\partial \mathcal{A}}{\partial t}(\boldsymbol{x}, \boldsymbol{k}, t) = -\omega'(k) \left(\hat{\boldsymbol{k}} \cdot \nabla_{\boldsymbol{x}}\right) \mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t),} \tag{10}$$

which tells us that our amplitude function $\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t)$ gets advected in space in the direction $\hat{\boldsymbol{k}}$ with speed $\omega'(k)$. Note that this speed corresponds exactly to the *group speed* which transports water wave energy [Johnson 1997] and wave packets [Jeschke and Wojtan 2017]. We provide a more detailed derivation of this equation in Appendix A.

Equation 10 is subject to boundary conditions:

$$\begin{aligned} &\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) = \mathcal{A}_{\text{ambient}}(\boldsymbol{x}, \boldsymbol{k}, t) && \text{on transmitting boundary} \\ &\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) = \mathcal{A}(\boldsymbol{x}, \boldsymbol{k}_{\text{reflect}}, t) && \text{on reflecting boundary} \end{aligned} \tag{11}$$

where $\mathcal{A}_{\text{ambient}}$ is an amplitude function defined outside of the domain, and $\boldsymbol{k}_{\text{reflect}} = \boldsymbol{k} - 2(\boldsymbol{n} \cdot \boldsymbol{k})\boldsymbol{n}$ is the wavevector reflected off the boundary with a normal $\boldsymbol{n}$. The reflecting boundary condition is based on how a planar wave reflects of a straight boundary.

Equations 7 and 10 are the key ingredients we need to simulate water waves. The actual interactive wave simulator advects $\mathcal{A}$ through space each time step using Equation 10, and the renderer then uses Equation 7 to reconstruct the water surface where needed. In this approach, $\mathcal{A}$ acts as the primary simulation variable, while η is used afterwards for reconstruction and visualization. ($\mathcal{A}$ is updated by the simulation, and then we compute η where needed by summing together many different waves using $\mathcal{A}$ as their amplitude.)

We note that the Gabor transform in Equation 5 is only used for the derivation, and there is no need to compute it. Similarly, the size of the Gaussian s does not appear in Equations 7 and 10, so it

is merely a theoretical construct that is useful for analysis but not actually a parameter in our numerical method. We explain how we discretize Equations 7 and 10 in Section 4.

3.3 Discussion

Accuracy of approximation. Equation 10 is a linearized approximation to the true evolution, which is valid only when $\boldsymbol{l}$ is close to $\boldsymbol{k}$. Thankfully, the Gaussian term in Equation 9 forces $\tilde{A}$ to be small whenever $\boldsymbol{l}$ is far from $\boldsymbol{k}$, so the errors in $\mathcal{A}$ caused by this approximation are exponentially small.

Furthermore, the reflecting boundary condition is based on geometrical optics and is valid only when the wavelength and the packet size (represented by the parameter s) are small compared to the boundary curvature. A more accurate handling of high curvature boundaries would require to take wave scattering effects into account, which we leave as future work and note that our simple reflecting boundary condition is sufficient for visual plausibility.

Low-frequency $\mathcal{A}$. We prove in Appendix B that the Fourier transform of $\mathcal{A}$ is essentially a low-pass-filtered version of the Fourier transform of η. In other words, we can reconstruct the wave height function η using only low-frequency variables. This frequency shift has important consequences in the eventual discretization of the method, because the Nyquist limit prevents us from discretizing the high-frequency η function directly without aliasing, while the discretization of $\mathcal{A}$ is possible even with coarse grid resolutions. Instead of the grid resolution imposing limits on the resolution of visible water waves, our algorithm will impose limits on the resolution of the *wave amplitudes*, which are not visualized directly and arguably more difficult to discriminate perceptually.

Phase Shifts. The Gabor transform allows us to discuss the water height in terms of *phase* as well as local amplitudes. The phase of each wavelet in Equation 7 is determined by $\boldsymbol{k} \cdot \boldsymbol{x} - \omega(k)t$. If we translate the entire function in space by replacing $\eta(\boldsymbol{x}, t)$ with $\eta(\boldsymbol{x}-\boldsymbol{y}, t)$, then we gain a phase shift of $-\boldsymbol{k} \cdot \boldsymbol{y}$ on the right hand side of the equation; a small shift in space can lead to a large phase shift if k is big. Consequently, although the amplitudes $\mathcal{A}$ are well-behaved even on coarse grids, the phases are still sensitive to Nyquist's limit and require either low frequency waves (small k) or very many samples of k for accurate reconstruction. Alternatively, if we do not require particular interference patterns, then we can avoid aliasing by adding a random initial phase shift to each wave [Cook 1986]:

$$\eta(\boldsymbol{x}, t) = \mathrm{Re} \int_{\mathbb{R}^2} \mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) e^{i\left(\boldsymbol{k} \cdot \boldsymbol{x} - \omega(k)t + \xi(\hat{\boldsymbol{k}})\right)} d\boldsymbol{k}, \tag{12}$$

where ξ is a random number for each wave direction that is fixed throughout time.

Hybrid simulation. Section 3.1 motivates our method as a hybrid between spectrum-based methods and PDE-based methods. This hybrid nature becomes more explicit if we examine the effect of the size of the Gaussian, s on our simulation variable $\mathcal{A}$:

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) \to A(\boldsymbol{k}) \qquad \text{as } s \to \infty \tag{13}$$

$$\frac{1}{s^2}\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) \to \eta_c(\boldsymbol{x}, t) e^{-i(\boldsymbol{k} \cdot \boldsymbol{x} - \omega(k)t)} \qquad \text{as } s \to 0 \tag{14}$$

When s approaches infinity and the Gaussian becomes a spatial constant, our method transitions to a traditional spectrum-based algorithm. On the other hand, when s goes to zero and the Gaussian becomes a Dirac delta function, our method computes a function similar to the water height, η. We never set s directly in our discretization, but it is essentially determined by the grid spacing, Δx. Thus, our method resembles a spectrum-based algorithm within a single grid cell, and it resembles a PDE-based discretization as we zoom out.

Relationship to Wave Packets. Although we used the Gabor transform to derive our approach in Section 3.2, we can alternatively derive this method from the water wave packets of Jeschke & Wojtan [2017], as we do in Appendix C. We show that Equation 7 emerges as the limiting behavior of an infinite number of wave packets, spanning all possible positions and wavevectors. From this perspective, Jeschke & Wojtan [2017] introduced one particular *Lagrangian* discretization of our continuum theory, which samples a small number of individual packets and tracks their propagation through space. Equation 10 represents an alternative *Eulerian* reference frame, which tracks the changes in the wave packet content at each point in space. We also note that, in Appendix C, the function $\mathcal{A}$ appears as a smooth average of all nearby wave packet amplitudes. This gives us further evidence that $\mathcal{A}$ is indeed a smooth function that varies slowly over space.

4 DISCRETIZATION

Section 3.2 introduced a new amplitude function $\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t)$, an equation for evolving it over time (Equation 10), and an equation for computing the water wave height (Equation 7). The remainder of this section explains how we discretize these ideas to efficiently simulate water waves.

4.1 Discretizing $\mathcal{A}$

The amplitude $\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t)$ is a function in 4 + 1 dimensions: two in space, two in wavevector, and one in time. We find it intuitive and computationally convenient to represent the wavevector in polar coordinates $\boldsymbol{k} = (k \cos\theta, k \sin\theta)$, where k is the magnitude of $\boldsymbol{k}$, and θ is the angle made by $\boldsymbol{k}$ and the x-axis. We represent $\mathcal{A}$ on a four-dimensional grid $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [0, 2\pi) \times [k_{\min}, k_{\max}]$, which spans two spatial coordinates x and y, the angular coordinate θ, and the wavevector coordinate k. We store samples of $\mathcal{A}$ on each of the nodes in this 4D grid, indexed by the coordinates a,b,c. We use the notation $\mathcal{A}_{abc}$ to represent the discrete amplitude sample of the wave at grid node position $\boldsymbol{x}_a = (x_a, y_a)$, traveling at angle θ_b, with wavenumber k_c. Figure 2 illustrates the grid used for our discretization.

We can now approximate $\mathcal{A}$ with a linear combination of basis functions, in the style of the finite element method:

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) = \sum_{a,b,c} C(\mathcal{A}_{abc}, t) \phi_a(\boldsymbol{x}) \vartheta_b(\theta) \psi_c(k). \tag{15}$$

where $\phi_a(\boldsymbol{x})$ is a basis function in position, $\vartheta_c(\theta)$ is a basis function in angle, $\psi_c(k)$ is a basis function in wavenumber coordinates, and $C(\mathcal{A}_{abc}, t)$ is a coefficient function weighting various values of

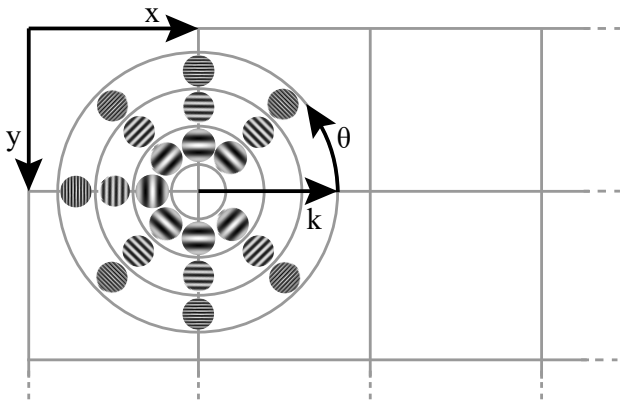


Fig. 2. A schematic of our 4D grid for discretizing $\mathcal{A}(x, y, \theta, k)$. Each spatial location (x, y) is equipped with a polar coordinate grid (k, θ). Coordinates $(x_a, y_a, \theta_b, k_c)$ are used to store the amplitude of a wavelet at position (x_a, y_a), angle θ_b, and spatial frequency k_c.

$\mathcal{A}_{abc}$. For example, piecewise-linear basis functions in all dimensions would set $C(\mathcal{A}_{abc}) = \mathcal{A}_{abc}$, giving us

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) = \sum_{a,b,c} \mathcal{A}_{abc}(t)\phi_a(\boldsymbol{x})\vartheta_b(\theta)\psi_c(k), \tag{16}$$

in which the basis functions are simply hat functions that weight the nearest values $\mathcal{A}_{abc}$.

The only practical restriction on the basis functions is that our reconstructed function should actually interpolate the discrete samples $\mathcal{A}_{abc}$ if we want operations like semi-Lagrangian advection to work as expected; in other words, $\mathcal{A}(\boldsymbol{x}_a, \theta_b, k_c, t) = \mathcal{A}_{abc}(t)$. The spatial basis function $\phi_a(\boldsymbol{x})$ simply interpolates $\mathcal{A}_{abc}$ over space, and we use standard piecewise-linear or piecewise-cubic basis functions here. The angular weighting $\vartheta_b(\theta)$ interpolates amplitudes across different traveling directions, and we use piecewise cubic basis functions here as well.

The wavenumber basis $\psi_c(k)$ has a special physical interpretation; it symbolizes the spectrum of the waves represented by amplitude $\mathcal{A}_{abc}$. If we continue with our wave packet analogy, then $\psi_c(k)$ describes the shape in frequency-space of the wave packet at position $\boldsymbol{x}$ traveling in direction θ with representative wavenumber k_c. Thus, a piecwise-constant $\psi_c(k)$ implies that the wave packet has a flat spectrum, with all wavenumbers similar to k_c having the exact same amplitude; a piecewise-*linear* $\psi_c(k)$ gives the packet a bit more of a localized wave packet-like shape with its peak at k_c; and a Gaussian $\psi_c(k)$ resembles the typical wave packet derivation. We are free to assign any wave spectrum we wish to this function, and, since we are modeling water waves in this paper, we found it appropriate to use an actual ocean wave spectrum for $\psi_c(k)$. Our examples set $\psi_c(k)$ equal to the directional spectrum in Equation 32 of [Horvath 2015], normalized such that $\mathcal{A}(\boldsymbol{x}_a, \theta_b, k_c, t) = \mathcal{A}_{abc}(t)$.

In practice, we found that we can use surprisingly coarse grids for this discretization. Our simulations use $\Theta_{\mathcal{A}} = 16$ samples for the wavevector angle, θ; finer discretizations did not increase the simulation quality. We use even fewer samples for discretizing the wavenumber, k; we typically only need $K_{\mathcal{A}} = 1$ sample to get the effects we desire, though we experiment with up to $K_{\mathcal{A}} = 4$ samples in Section 8. As mentioned above, $\mathcal{A}$ varies slowly over space, so we do not require much spatial resolution either. In our implementation we allocate $X_{\mathcal{A}} = 4096$ grid cells for each spatial dimension, which defines a grid cell spacing of approximately one meter.

Lastly, the simulations in this paper use real-valued $\mathcal{A}$ functions as initial conditions. This function then stays real for all time according to Equation 10, so our implementation does not bother to store complex $\mathcal{A}_{abc}$ coefficients.

4.2 Discretizing Advection

Once we have discretized all of the wavevectors $\boldsymbol{k}$, Equation 10 becomes a small number of independent scalar advection equations in space—one for each sample of $\boldsymbol{k}$. We numerically integrate each of these equations in parallel using the unconditionally stable semi-Lagrangian advection with slope-limited cubic spatial interpolation proposed by Fedkiw et al. [Fedkiw et al. 2001]:

$$\mathcal{A}_{abc}(t + \Delta t) = \mathcal{A}_{bc}(\boldsymbol{x}_a - \Delta t\omega'(k_c)\hat{\boldsymbol{k}}_b, t), \tag{17}$$

where $\mathcal{A}_{bc}$ is the interpolation of the discretized $\mathcal{A}$ with fixed angle θ_b and wavenumber k_c, $\hat{\boldsymbol{k}}_b = (\cos\theta_b, \sin\theta_b)$ is the wave direction determined by angle θ_b, and $\boldsymbol{x}_a = (x_a, y_a)$.

Whenever a semi-Lagrangian ray leaves the simulation domain, we apply the relevant boundary condition from Equation 11 by using a different value on the right hand side of Equation 17. For transmitting boundaries, we assign a procedural $\mathcal{A}_{\text{ambient}}$ function describing the ambient ocean behavior. For reflecting boundaries, we reflect the semi-Lagrangian ray off the boundary to get a new position $\boldsymbol{x}_{\text{reflect}}$ and new direction $\boldsymbol{k}_{\text{reflect}}$, and we update $\mathcal{A}$ with the reflected function value $\mathcal{A}(\boldsymbol{x}_{\text{reflect}}, \boldsymbol{k}_{\text{reflect}}, t)$.

Amplitude Spreading. Amplitudes with different angle θ_b or wavenumber k_c have either different travel direction or speed. Over time these amplitudes separate from each other and a nice initial state can turn into bunch of separated amplitude blobs as depicted on the left part of Figure 3. This is a problem of discretizing Equation 10 with finitely many wavevectors and we deal with it by adding two diffusion terms:

$$\frac{\partial \mathcal{A}}{\partial t} = -\omega'(k)\left(\hat{\boldsymbol{k}} \cdot \nabla_{\boldsymbol{x}}\right)\mathcal{A} + \delta\left(\hat{\boldsymbol{k}} \cdot \nabla\right)^2 \mathcal{A} + \gamma\frac{\partial^2 \mathcal{A}}{\partial \theta^2}, \tag{18}$$

where δ controls diffusion in the direction of travel (mimicking how wave packets stretch out due to dispersion), and γ controls diffusion in the travel angle (mimicking how wave packets spread out tangentially as they radiate from a source). Other diffusive terms are possible, but we found that the above two terms work really well. Figure 3 illustrates the effect of our angular diffusion term.

We apply operator splitting to solve this advection-diffusion equation numerically: the advection is solved by semi-Lagrangian advection as described above, and the diffusion is discretized with second order finite differencing in space and forward Euler in time. In our examples, we set the diffusion parameters relative to the advection speed ω', the spatial resolution Δx, the wavenumber resolution Δk and the angular resolution $\Delta\theta$: $\gamma = 0.025\omega'(k)\Delta\theta^2/\Delta x$ and $\delta = 10^{-5}\Delta x^2\Delta k^2\,|\omega''(k)|$.

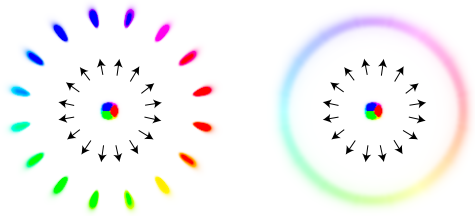


Fig. 3. The effects of our diffusion term are most obvious in this isolated scenario where waves radiate from a point. Without angular diffusion (left), the wavefronts eventually separate. Adding angular diffusion (right) allows the wavefronts to stay connected over time.

4.3 Height field evaluation

To calculate the actual water height we numerically evaluate the integral in Equation 12. We evaluate the wavenumber in polar coordinates

$$\eta(\boldsymbol{x}, t) = \mathrm{Re} \int_0^{2\pi} \int_0^{\infty} \mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) e^{i(\boldsymbol{k}\cdot\boldsymbol{x} + \xi(\hat{\boldsymbol{k}}) - \omega(k)t)}\, k\, dk\, d\theta \quad (19)$$

and plug in the discretized form of $\mathcal{A}$ to get:

$$\eta(\boldsymbol{x}, t) = \int_0^{2\pi} \sum_{a,b,c} C(\mathcal{A}_{abc}, t)\phi_a(\boldsymbol{x})\vartheta_b(\theta)\overline{\Psi}_c\left(\hat{\boldsymbol{k}}\cdot\boldsymbol{x} + \xi(\hat{\boldsymbol{k}}), t\right) d\theta \quad (20)$$

$$\overline{\Psi}_c(p, t) = \int_0^{\infty} \psi_c(k)\cos(kp - \omega(k)t)\; k\, dk, \quad (21)$$

Now we can see more explicitly how the basis function $\psi_c(k)$ relates to the spectrum associated with amplitude $\mathcal{A}_{abc}$; Equation 21 treats $\psi_c(k)$ as the amplitude function for each wavelength k in the heightfield evaluation.

To use the above formulas, we first pre-compute the function $\overline{\Psi}_c(p, t)$ at the beginning of each time step by evaluating it at a number of discrete sample points p_i and store it in a 1D texture, which we call a "profile buffer". This pre-computation allows us to turn this costly integral evaluation into a simple texture lookup for the rest of the time step. After the pre-computation, we can evaluate the water height at any point in space by approximating the one-dimensional integral in Equation 20 with a summation over several sampled angles. To create our examples, we sum over $K_\eta = 400$ wavenumbers for the pre-computation in Equation 21, and we store the resulting function $\overline{\Psi}_c$ in a 1D texture of size $N = 4096$. We sum over $\Theta_\eta = 120$ angles to evaluate η at each point in space. Note that the number of directions and wavenumbers for discretizing $\mathcal{A}$ compared to η are largely independent. We chose these parameters empirically based on the quality of our results, and we list them in Table 1.

Our examples also extend the water height field to mimic trochoidal Gerstner waves [Tessendorf 2004b] by computing horizontal displacements in addition to the vertical ones in the profile buffer $\overline{\Psi}_c(p, t)$. We then include these horizontal displacements in the final summation of η.

Table 1. Parameters used to create the examples in our paper.

Variable	Value	Description
$K_{\mathcal{A}}$	1	No. of k samples for discretizing $\mathcal{A}$
K_η	400	No. of k samples for integrating η
$\Theta_{\mathcal{A}}$	16	No. of θ samples for discretizing $\mathcal{A}$
Θ_η	120	No. of θ samples for integrating η
$X_{\mathcal{A}}$	4096^2	No. of spatial grid samples for discretizing $\mathcal{A}$
N	4096	No. of 1D texture samples in profile buffer

The pre-computation of the profile buffer in Equation 21 is largely responsible for the performance of our method. For comparison, the evaluating η at all points in space using the naive 2D integral in Equation 19 costs $O(K_\eta\Theta_\eta X_{\mathcal{A}}^2)$ operations, where $X_{\mathcal{A}}^2$ is the total number of spatial locations where η is computed. Our speedup using the profile buffer reduces this computation by two orders of magnitude to $O(K_\eta + \Theta_\eta X_{\mathcal{A}}^2)$. Distributing this computation over G GPU cores reduces the cost further to $O(K_\eta + \Theta_\eta X_{\mathcal{A}}^2/G)$. In practice, the introduction of the profile buffer in our implementation raised the frame rate from 1.8 to 275 – a speed-up factor of 233 for evaluating η.

5 ALGORITHM SUMMARY

This section gives an overview of the steps necessary to implement our algorithm. Our project webpage[1] also provides example code for a straightforward (CPU-only) implementation of the algorithm as well as an executable file which demonstrates our GPU-optimized implementation.

The main goal of our algorithm is to update the amplitudes $\mathcal{A}$ (Equation 10) and use them to visualize the water height η (Equation 7). We note that almost all of the physics simulation happens in the computation of the amplitudes, and that $\mathcal{A}$ is never directly visualized. On the other hand, the comptuation of η is relatively light (thanks to the pre-computed profile buffer), and its visualization is almost entirely responsible for the apparent detail in the wave simulation. To take advantage of this disparity, we compute $\mathcal{A}$ on coarse grids, and we compute η on a viewer-dependent adaptively-refined detailed mesh. Specifically, our GPU-optimized version uses hardware tesselation [Nießner et al. 2016] to compute an adaptive triangle mesh with vertex positions determined by η, and it computes the surface normals in a pixel shader using the analytic spatial derivatives of η.

We divide our algorithm into a function `TimeStep` that does some pre-computation work once every time step, and a function `WaterHeight` that needs to be computed on-demand for each node of the finely-sampled grid and each pixel. `TimeStep` mainly solves the evolution equation 18 by splitting it into two parts: `AdvectionStep`, which computes the semi-Lagrangian advection in

[1] http://visualcomputing.ist.ac.at/publications/2018/WSW/

Section 4.2, and `WavevectorDiffusion`, which computes the amplitude spreading. It finishes with the function `PrecomputeProfileBuffers` which precomputes the one dimensional water wave profile buffers $\overline{\Psi}_c(p,t)$ which are used for water height evaluation (Section 4.3). The `WaterHeight` function numerically evaluates Equation 20 with a weighted sum of 1D wave profiles at different angles, as in Section 4.3. Please see Algorithm 1 for pseudocode.

Algorithm 1 Pseudocode for the algorithms used in our paper

```
function TimeStep(t)
    AdvectionStep(t)
    WavevectorDiffusion(t)
    Ψ̄ ← PrecomputeProfileBuffers(t)
end function

function WaterHeight(x,t)
    η ← 0
    for b ← 1, Θ_η do
        θ_b ← (2π/Θ_η) b
        k̂ ← (cos θ_b, sin θ_b)
        p ← k̂ · x + rand(b)
        for c ← 1, K_η do
            η ← η + A(x, k_c k̂) * Ψ̄_c(p, t)
        end for
    end for
end function
```

6 EXTENSIONS

Once we have our basic water wave solver in place, we can extend it in some interesting ways. This section introduces dissipation, a strategy for pre-computing the wave physics for rapid wave playback, and a method for interacting with water waves using solid-fluid coupling.

6.1 Dissipation

In nature, wave energy is lost due to viscosity, splashes and so on. In order to reproduce this behaviour, we add the same dissipation term as Jeschke et al. [2017] to Equation 18:

$$\frac{\partial \mathcal{A}}{\partial t} = \cdots - 2\nu k^2 \mathcal{A} - \frac{1}{2}\nu \left(\frac{\omega(k)}{2\nu}\right)^{\frac{1}{2}} k\mathcal{A} \tag{22}$$

The first term models dissipation due to water viscosity ν ($10^{-6} m/s$), which affects mostly waves with small wavelengths. The second term accounts for surface contamination such as oil, dirt or algae and it affects larger waves as well.

6.2 Pre-computing wave motions

The method of Jeschke & Wojtan [2015] computes steady-state wave motions by pre-computing the paths of multiple wavefronts, storing the phase functions on an adaptive triangle mesh, and proposing a phase interpolation scheme to recover the final waves at runtime. We can produce similar results with our simulator by computing wave motion until the amplitudes reach a steady state, and then storing the $\mathcal{A}$ functions as static textures. We can then interpolate these $\mathcal{A}$ textures at any point in space and use them to efficiently compute wave heights at run-time. This pre-computation of $\mathcal{A}$ leads to a considerable speed-up over running the full simulation (4.7× speedup from ≈ 60fps to ≈ 280fps in this example). Our method is arguably more efficient than Lagrangian wavefront tracking (though the methods have very different numerical parameters and the errors behave differently), and the run-time behavior is more convenient and load-balanced for graphics hardware than an adaptive triangle mesh.

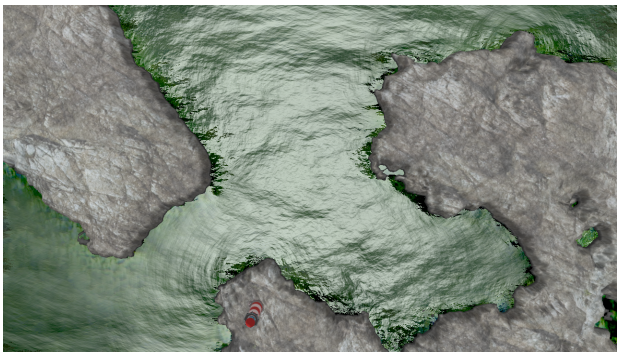


Fig. 4. The wave amplitudes $\mathcal{A}$ in this animation were pre-calculated by a wind-driven simulation, but the wave heights η are computed at runtime. Notice the boundary effects where waves reflect and diffract, as well as the shadowing effect where the island blocks the wind.

Figure 4 shows an example scene that was pre-computed using this technique. To make this result, we modeled wind effects by adding a source term $\mathcal{S}(\boldsymbol{x}, \boldsymbol{k}, t)$ to Equation 18. Local wind effects like this are difficult for wavefront tracking methods which require the explicit creation of coherent wavefronts as initial conditions, but they are easy for our Eulerian method. However, unlike [Jeschke and Wojtan 2015] our simulator does not yet handle shallow water effects due to a spatially varying dispersion relation ω, and our jittered sampling strategy makes it difficult to precisely control the wavefront phases.

6.3 Solid-Fluid Coupling

We implemented some elementary coupling between our fluid simulator and rigid bodies. We do this coupling by adding forces to the rigid body from the fluid, and adding waves to the fluid from the rigid body each time step.

To make the waves influence the rigid bodies, we compute a buoyancy force by approximating the shape of the submerged volume of the rigid body as a cylinder with volume $V = \pi r^2(\eta - r - h)$, where r is the radius of the body and h is the height of its center of mass. The buoyancy force is then $-\rho V \boldsymbol{g}$, where ρ is the density of the fluid and $\boldsymbol{g}$ is the gravity vector. We integrate this force using backward Euler integration to ensure numerical stability.

To add waves to the rigid body simulation, we calculate the change in energy of the rigid body caused by the water ΔE_{RB}, where the rigid body's energy E_{RB} is equal to $mv^2/2 + mgh$, with body mass m, velocity magnitude v, and gravity magnitude g. Linear wave theory

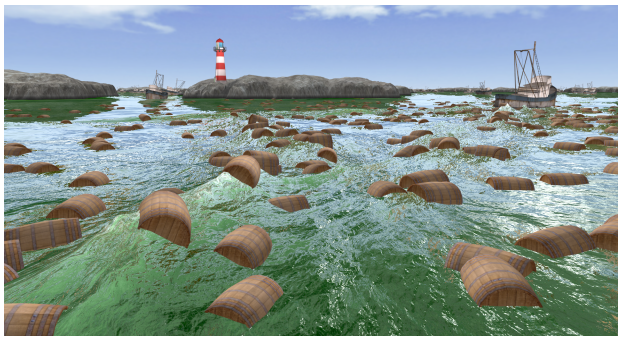


Fig. 5. The barrels receive buoyancy forces and impose new waves at the same time.

tells us that water wave energy in deep water is proportional to the squared amplitude:

$$E_{\text{fluid}}(k) = \frac{1}{2}\rho g\left(\mathcal{A}(k)\right)^2 \tag{23}$$

and we use our localized amplitudes to confine the wave energy to a region of space:

$$E_{\text{fluid}}(\boldsymbol{x}, \boldsymbol{k}, t) \approx \frac{1}{2}\rho g\left(\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t)\right)^2 . \tag{24}$$

To approximately conserve energy, we must set $\Delta E_{\text{fluid}} = -\Delta E_{\text{RB}}$. Assuming this change in energy is distributed equally among all wavevectors, we can then compute the new amplitudes $\mathcal{A}$ at the location of the rigid body $\boldsymbol{x}_{\text{RB}}$:

$$\mathcal{A}^{\text{new}}(\boldsymbol{x}_{\text{RB}}, \boldsymbol{k}, t) \approx \sqrt{\frac{2}{\rho g}\left(E_{\text{fluid}}(\boldsymbol{x}_{\text{RB}}, \boldsymbol{k}, t) - \frac{\Delta E_{\text{RB}}}{K_{\mathcal{A}}\Theta_{\mathcal{A}}}\right)}, \tag{25}$$

where the product $K_{\mathcal{A}}\Theta_{\mathcal{A}}$ is the number of discrete wavevectors in our discretization.

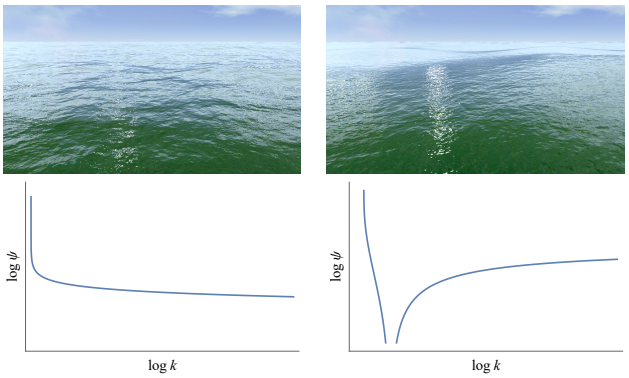
To summarize, our rigid body coupling is executed each time step by adding buoyancy forces to each body, calculating the change in energy of the body, and then converting that change in energy into waves at the location of the rigid body. Figure 5 uses this technique in an example with numerous floating boxes. This heuristic coupling strategy works well for displaying basic solid-fluid interactions, and we leave a more careful treatment for future work.

7 ARTISTIC CONTROL

The method we have presented so far focused on an efficient approximation for the physical motion of water waves. However, we can selectively replace various steps in our algorithm with user-defined procedures, in order to add artistic effects at the expense of physical realism.

7.1 Selecting the basis function $\psi(k)$

The basis function $\psi(k)$ controls the wave spectrum which is visualized in the final results. The spectrum can either be determined by physics or tuned by hand to create more stylized results. Figure 6 shows how changing this function affects the visualized wave heights. Regardless of the chosen spectrum, the waves will travel at the correct phase speed due to the dispersion relation in Equation 21.



Phillips spectrum User-defined spectrum

Fig. 6. The effect of varying the wavenumber basis function $\psi(k)$: We produce different wave styles by setting ψ to a standard Phillips spectrum (left) and a user-defined spectrum (right).

7.2 Manually overriding $\mathcal{A}$

Instead of using physical equations to compute the amplitude textures $\mathcal{A}$, we can explicitly create or modify them with procedural functions, or with a novel amplitude-painting interface. In our video, we show that we can procedurally create reflection effects by artificially amplifying waves traveling perpendicular to an obstacle's surface, with an amplification factor based on distance to the obstacle. We can also use a painting interface, as shown in Figure 7.

We note that hand-tuning these amplitudes will be physically incorrect in the sense that the group speeds are set to zero instead of determined by the dispersion relation. However, the phase speeds are still physically correct due to the dispersion relation in Equation 21. The result is that the waves themselves travel at the correct speed, but the wave *groups* do not. We found this indirect physical inaccuracy a bit more difficult to perceive, and so we believe that such an amplitude override technique might be a useful artistic tool.

8 RESULTS

We show numerous results created by our method in our supplemental videos. To illustrate the large scale and interactive nature of

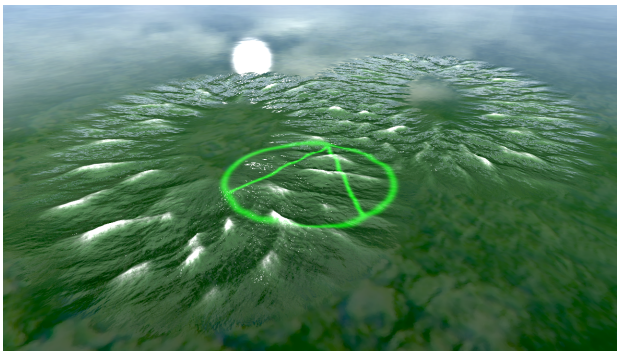


Fig. 7. Overriding $\mathcal{A}$ with a real-time wave-painting interface to make waves process along an ∞-shaped path.

Table 2. Performance breakdown for a single frame of animation.

Algorithm Component	Timing	% of Total
Updating $\mathcal{A}$	8.54ms	51%
Computing η	4.16ms	25%
Miscellaneous rendering and unrelated overhead	3.97ms	24%

our results, we show a vast 4km × 4km sea interacting with islands, floating barrels, actively moving boats, and a user-controlled jet-ski. Both the simulation and the heightfield evaluation are computed in parallel on the GPU in each time step. We provide a supplemental document that describes relevant implementation details for both parts. Our laptop with a NVIDIA Geforce GTX 1070 GPU achieves an average frame rate of 60fps with the parameters in Table 1, and this paper includes an interactive demo of our method which recreates this example. Table 2 displays the timing breakdown for an average frame of this animation; note that the timing for the computation of η depends on the number of pixels occupied by by waves and may vary slightly.

Varying these parameters has different effects on the visual results and performance of our method, and we explore each of them in our supplementary video. The number of $\mathcal{A}$ samples in our simulation depends linearly on the resolution of our 4D $X_{\mathcal{A}} \times X_{\mathcal{A}} \times \Theta_{\mathcal{A}} \times K_{\mathcal{A}}$ simulation grid, so doubling the resolution of any dimension will roughly increase the memory and the runtime by a factor of 2. Increasing the spatial resolution $X_{\mathcal{A}}$ will allow the wavefronts to exhibit a higher curvature, allowing more detailed interactions with highly curved boundaries. Figure 8 shows the effect of $X_{\mathcal{A}}$ on the simulation quality. Increasing the angular resolution $\Theta_{\mathcal{A}}$ allows a more precise behavior in each direction. Increasing the wavenumber resolution $K_{\mathcal{A}}$ allows more detailed dispersion of wave groups (different amplitude groups travel at different speeds). We show an example with $K_{\mathcal{A}} = 4$ simulated wave groups in Figure 9 and in our video, which shows more accurate wave group dispersion but roughly quadruples the run time (drops the frame rate from 70fps to 20fps).

Note that none of these resolution parameters affect the resolution of the waves themselves; they only affect the resolution of the wave *groups*, and thus induce higher-order indirect effects like curvature and speed of the wave groups, instead of affecting more visible cues like the frequency or speed of the wave crests. Instead, the frequency of the waves is controlled by the resolution of the heightfield evaluation, $\eta(\boldsymbol{x}, t)$, and the relative speed of the waves is fixed by the dispersion relation ω.

For discretizing $\mathcal{A}$ we chose a spatial resolution of $X_{\mathcal{A}} = 4096$ for each dimension because it maps well to the GPU and allows an exceptionally large simulation domain. Many of the up-close interactions in our video have an effective resolution of approximately 10^2 grid cells on the screen at a time. We chose $\Theta_{\mathcal{A}} = 16$ wave directions for the simulation because it maps well to the GPU, and because fewer samples showed some directional bias artifacts when visualizing the $\mathcal{A}$ function directly. We could not tell much difference if we increase the angular resolution to 32. We chose only $K_{\mathcal{A}} = 1 - 4$ wavenumber samples because we did not think the accurate simulation of wave group dispersion was necessary for visual effects.

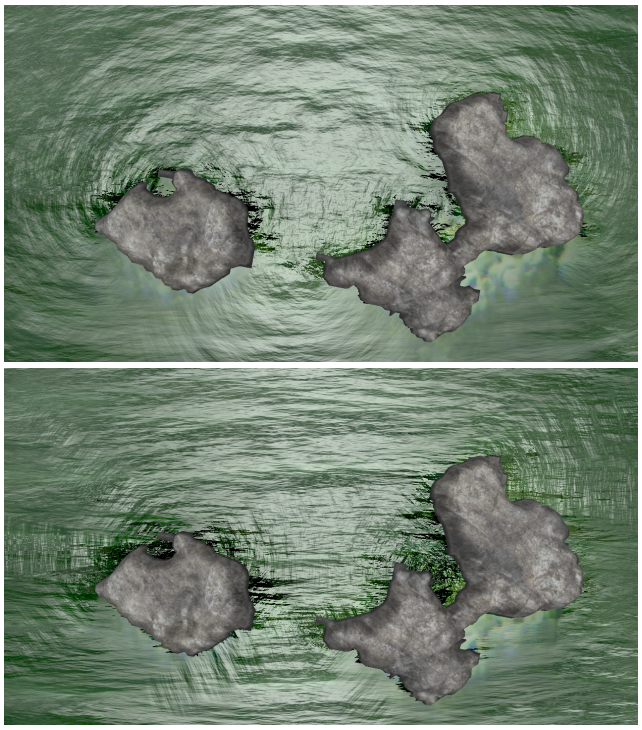


Fig. 8. A detailed simulation (top) and one with the spatial resolution $X_{\mathcal{A}}$ reduced by 4× in each dimension (bottom). Reducing $X_{\mathcal{A}}$ does not affect the resolution of the waves themselves, but it affects the resolution of the wave *groups*. At this low resolution, reflected and diffracted waves cannot resolve the detailed boundaries properly, and wavefront curvature is reduced.

The resolution of η (Equation 20), however, has a direct effect on the visual results. Reducing the number of spatial samples will reduce the highest visible frequency, using only a small number of θ samples will introduce lattice-like artifacts caused by waves appearing as perfectly aligned, and using only a few k samples will remove visual frequencies from the final wave visualization.

Section 4.2 introduces an ad-hoc parameter for the angular amplitude diffusion. Large diffusion rates cause the amplitudes to blur all directions together quickly, making the waves more isotropic; small diffusion rates cause the wave packets to separate from each other, as illustrated in Figure 3. The effects of this parameter are more evident near circular wave sources, where amplitudes are still large and exhibit higher curvature. The amplitudes naturally drop off further away from sources like this, making diffusion effects difficult to notice.

The performance of our method comes from a few sources. First, the fact that $\mathcal{A}$ is low resolution allows us to discretize it on a coarse grid, so we don't need an expensive simulation of $\mathcal{A}$ to get detailed visual results. We can exploit this coarse grid by either using a huge simulation domain (as in the above example), or by using very few degrees of freedom to make the simulation faster. Next, the pre-computed profile buffer $\overline{\Psi}$ saves us two orders of magnitude

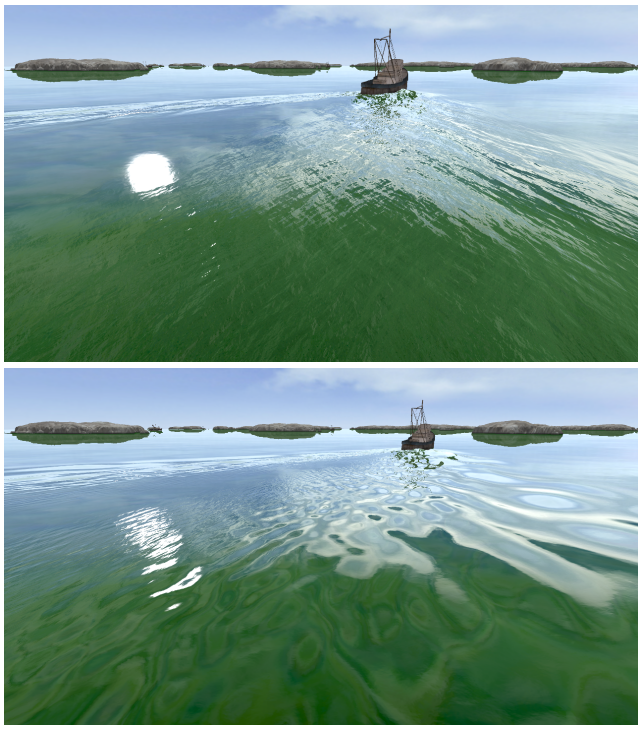


Fig. 9. Using a single wavenumber $K_{\mathcal{A}} = 1$ (top) results in a uniform distribution of small and large waves in this boat wake. By contrast, when using four wavenumbers $K_{\mathcal{A}} = 4$ (bottom) we can notice how larger waves on the right move ahead of smaller ones on the left.

in computation by reducing a 2D integral to a 1D integral with a texture lookup. Lastly, both the simulation and the wave height evaluation are embarrassingly parallel operations spread out among many points in space, so they greatly benefit from GPU acceleration.

9 DISCUSSION

This paper proposes a novel wavelet-based discretization for animating water waves. As it is based on linear wave theory, it can only approximate the correct behavior for waves with small amplitudes and is incapable of capturing any non-linear effects. The main dynamical equation, Equation 10 or 18, is a linear differential equation in $\mathcal{A}$. Our method handles non-heightfield displacement effects like Biesel and Gerstner waves, but there is no direct way for it to handle complex non-linear phenomena like breaking waves or topology changes like splashes.

This approach de-couples the resolution of the visualized waves from the resolution of the simulation. Through a novel Gabor transformation, we are able to keep the simulation resolution much lower than the resolution of the heightfield which is ultimately visualized. Thus, this approach can animate very high frequency waves without the typical complications relating to excessive computation, aliasing, or simulation stability.

Compared to Eulerian height field-based simulations, our method stores 4096^2 (spatial resolution) $*16$ (wave vector resolution) samples for our 4 km by 4 km scene. A height field storing the same number of samples would have a grid cell spacing of 25 cm, even ignoring that it needs to store 2 values per grid cell. Following the Nyquist theorem, the smallest possible wavelength would be 0.5 m. By comparison, we animate wavelengths down to 2 cm.

Compared to wave packets [Jeschke and Wojtan 2017], neighboring overlapping wave packets cause massive pixel overdraw during rendering, which significantly reduces performance and there is no easy way to fix this problem. To illustrate the performance difference, a single boat wake takes from 2 up to 6 Mill wave packets, and it renders at 2 to 0.5 FPS respectively. By contrast, our method simulates and renders 1000 boat wakes at 60 FPS on the same hardware, and it naturally offers constant computational cost, i.e., it does not depend on the number of waves being simulated. However, the boat wake of wave packets is physically more accurate as phases of individual waves are explicitly controlled. As a guideline, wave packets should be used if control over wave phase (for perfectly circular ripples for example) is crucial and the number of packets is not too high. Surface wavelets are clearly the better choice for interactive water simulations even at medium scales where plausibility is more important than physical accuracy.

Our method can efficiently simulate the aggregate motion of high-frequency water waves, even with a low resolution simulation. However, as discussed in Section 3.3, low-resolution simulations give up the ability to precisely control the phase of each wave. Consequently, it is difficult to simulate phenomena that depend upon coherent phases, like the perfect circular wavefronts emitted from raindrops, without increasing the simulation resolution. Similarly, many familiar wake patterns that depend on constructive interference between coherent phases [Jeschke and Wojtan 2017; Thomson 1891] are impractical to replicate with our method. At low resolutions, our proposed jittered phases are better suited for noisy wave sources like chaotic splashes, wind, and large floating objects. We hope future research can remove this connection between wave phase coherence and simulation resolution.

Our current implementation uses the deep water dispersion relation. In the future, we would like to extend this work to handle a more general depth-dependent dispersion relation, which should create additional refractive effects near shallow water.

Overall, we believe that our approach of simulating spatially-dependent amplitudes presents an interesting twist on water wave simulation. This new direction introduces unique challenges, like increased dimensionality and an interesting link between phase and resolution. At the same time, it makes significant progress on outstanding problems in the field of physics-based animation: it introduces novel methods for artistic control, it permits extremely large simulation domains, and it enables interactive animations with fine spatial resolutions.

ACKNOWLEDGMENTS

We wish to thank the anonymous reviewers and the members of the Visual Computing Group at IST Austria for their valuable feedback. The 3D models used in our examples were created by Reiner Prokein. This research was supported by the Scientific Service Units (SSU) of IST Austria through resources provided by Scientific Computing. This project has received funding from the European Research

Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreements No 638176 and Marie Skłodowska-Curie Grant Agreement No. 665385.

REFERENCES

Roland Angst, Nils Thuerey, Mario Botsch, and Markus Gross. 2008. Robust and efficient wave simulations on deforming meshes. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 1895–1900.

Robert Bridson. 2015. *Fluid simulation for computer graphics.* CRC Press.

José A. Canabal, David Miraut, Nils Thuerey, Theodore Kim, Javier Portilla, and Miguel A. Otaduy. 2016. Dispersion Kernels for Water Wave Simulation. *ACM Trans. Graph.* 35, 6, Article 202 (Nov. 2016), 10 pages.

Stephen Chenney. 2004. Flow tiles. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation.* Eurographics Association, 233–242.

Robert L. Cook. 1986. Stochastic Sampling in Computer Graphics. *ACM Trans. Graph.* 5, 1 (Jan. 1986), 51–72.

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual Simulation of Smoke. In *Proceedings of SIGGRAPH 2001 (Computer Graphics Proceedings, Annual Conference Series)*, Eugene Fiume (Ed.). ACM, 15–22.

Dennis Gabor. 1946. Theory of communication. Part 1: The analysis of information. *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering* 93, 26 (1946), 429–441.

Robert Geist, Christopher Corsi, Jerry Tessendorf, and James Westall. 2010. Lattice-boltzmann water waves. In *Advances in visual Computing*. Springer, 74–85.

Damien Hinsinger, Fabrice Neyret, and Marie-Paule Cani. 2002. Interactive animation of ocean waves. In *Proc. ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* 161–166.

Christopher J Horvath. 2015. Empirical directional wave spectra for computer graphics. In *Proceedings of the 2015 Symposium on Digital Production.* ACM, 29–39.

Stefan Jeschke and Chris Wojtan. 2015. Water Wave Animation via Wavefront Parameter Interpolation. *ACM Trans. Graph.* 34, 3, Article 27 (May 2015), 14 pages.

Stefan Jeschke and Chris Wojtan. 2017. Water Wave Packets. *ACM Trans. Graph.* 36, 4, Article 103 (2017), 12 pages.

R.S. Johnson. 1997. *A modern introduction to the mathematical theory of water waves.* Vol. 19. Cambridge university press.

M. Kass and G. Miller. 1990. Rapid, stable fluid dynamics for computer graphics. In *Computer Graphics*, Vol. 24. 49–57.

Theodore Kim, Jerry Tessendorf, and Nils Thuerey. 2013. Closest Point Turbulence for Liquid Surfaces. *ACM Trans. Graph.* 32, 2, Article 15 (April 2013), 13 pages.

Jörn Loviscach. 2002. A convolution-based algorithm for animated water waves. In *Eurographics*, Vol. 2. 381–389.

Pierre-Luc Manteaux, Ulysse Vimont, Chris Wojtan, Damien Rohmer, and Marie-Paule Cani. 2016. Space-time sculpting of liquid animation. In *Proceedings of the 9th International Conference on Motion in Games.* ACM, 61–71.

Gary A Mastin, Peter A Watterberg, and John F Mareda. 1987. Fourier synthesis of ocean scenes. *Computer Graphics and Applications, IEEE* 7, 3 (1987), 16–23.

A. McNamara, A. Treuille, Z. Popović, and J. Stam. 2004. Fluid control using the adjoint method. In *ACM Trans. Graph.*, Vol. 23. ACM, 449–456.

Michael B Nielsen, Andreas Söderström, and Robert Bridson. 2013. Synthesizing waves from animated height fields. *ACM Trans. Graph.* 32, 1 (2013), 2.

M. Nießner, B. Keinert, M. Fisher, M. Stamminger, C. Loop, and H. Schäfer. 2016. Real-Time Rendering Techniques with Hardware Tessellation. *Computer Graphics Forum* 35, 1 (2016), 113–137.

Björn Ottosson. 2011. *Real-time Interactive Water Waves.* Master's thesis. KTH, Sweden.

Zherong Pan, Jin Huang, Yiying Tong, Changxi Zheng, and Hujun Bao. 2013. Interactive localized liquid motion editing. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 184.

K. Raveendran, N. Thuerey, C. Wojtan, and G. Turk. 2012. Controlling Liquids Using Meshes. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation.* 255–264.

Karthik Raveendran, Chris Wojtan, Nils Thuerey, and Greg Turk. 2014. Blending liquids. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 137.

Bruce Schachter. 1980. Long crested wave models. *Computer Graphics and Image Processing* 12, 2 (1980), 187–201.

Claude Elwood Shannon. 1949. Communication in the presence of noise. *Proceedings of the IRE* 37, 1 (1949), 10–21.

Lin Shi and Yizhou Yu. 2005. Taming liquids for rapidly changing targets. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation.* ACM, 229–236.

Jerry Tessendorf. 2004a. Interactive water surfaces. *Game Programming Gems* 4 (2004), 265–274.

Jerry Tessendorf. 2004b. Simulating ocean water. *ACM SIGGRAPH Courses* (2004).

Jerry Tessendorf. 2014. *eWave: Using an Exponential Solver on the iWave Problem.* . Technical Note.

William "Lord Kelvin" Thomson. 1891. *Popular lectures and addresses.* Vol. 3. Macmillan London. 481–8 pages.

Nils Thuerey. 2016. Interpolations of Smoke and Liquid Simulations. *ACM Transactions on Graphics (TOG)* 36, 1 (2016), 3.

N. Thuerey, R. Keiser, M. Pauly, and U. Rüde. 2009. Detail-preserving fluid control. *Graphical Models* 71, 6 (2009), 221–228.

N. Thuerey, C. Wojtan, M. Gross, and G. Turk. 2010. A multiscale approach to mesh-based surface tension flows. *ACM Trans. Graph.* 29, 4 (2010), 48.

Sheng Yang, Xiaowei He, Huamin Wang, Sheng Li, Guoping Wang, Enhua Wu, and Kun Zhou. 2016. Enriching SPH simulation by approximate capillary waves. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* Eurographics Association, 29–36.

Jihun Yu, Chris Wojtan, Greg Turk, and Chee Yap. 2012. Explicit Mesh Surfaces for Particle Based Fluids. *EUROGRAPHICS 2012* 30 (2012), 41–48.

Cem Yuksel, Donald H. House, and John Keyser. 2007. Wave particles. In *Proc. SIGGRAPH.* 99.

A AMPLITUDE DYNAMICS

Here we provide a more detailed derivation of the amplitude evolution Equation 10. As we mentioned in Section 3.2, the evolution of $\mathcal{A}$ is fully determined by the integral Equation 8 which we use to derive the amplitude evolution.

The first step is to linearize the dispersion relation $\omega(l)$ in Equation 8 around the point $\boldsymbol{k}$, i.e. $\omega(l) \approx \omega(k) + \omega'(k)\hat{\boldsymbol{k}} \cdot (\boldsymbol{l} - \boldsymbol{k})$. Equation 8 becomes:

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) = \int_{\mathbb{R}^2} \tilde{A}(\boldsymbol{l}, \boldsymbol{k}) e^{i\left((\boldsymbol{l}-\boldsymbol{k})\cdot\boldsymbol{x} - \omega'(k)\hat{\boldsymbol{k}}\cdot(\boldsymbol{l}-\boldsymbol{k})t\right)} \, d\boldsymbol{l}. \tag{26}$$

Taking derivatives in time and space yields:

$$\frac{\partial \mathcal{A}}{\partial t} = - i\omega'(k)\hat{\boldsymbol{k}} \cdot \int_{\mathbb{R}^2} (\boldsymbol{l} - \boldsymbol{k}) \, \tilde{A}(\boldsymbol{l}, \boldsymbol{k}) e^{i\left((\boldsymbol{l}-\boldsymbol{k})\cdot\boldsymbol{x} - \omega'(k)\hat{\boldsymbol{k}}\cdot(\boldsymbol{l}-\boldsymbol{k})t\right)} \, d\boldsymbol{l}, \tag{27}$$

$$\nabla_{\boldsymbol{x}} \mathcal{A} = i \int_{\mathbb{R}^2} (\boldsymbol{l} - \boldsymbol{k}) \, \tilde{A}(\boldsymbol{l}, \boldsymbol{k}) e^{i\left((\boldsymbol{l}-\boldsymbol{k})\cdot\boldsymbol{x} - \omega'(k)\hat{\boldsymbol{k}}\cdot(\boldsymbol{l}-\boldsymbol{k})t\right)} \, d\boldsymbol{l}. \tag{28}$$

We see that the time change of $\mathcal{A}$ is just a $-\omega'(k)\hat{\boldsymbol{k}}$ multiple of the spatial gradient of $\mathcal{A}$. Therefore we obtained the evolution equation:

$$\frac{\partial A}{\partial t} = -\omega'(k)\hat{\boldsymbol{k}} \cdot \nabla_{\boldsymbol{x}} \mathcal{A}. \tag{29}$$

B $\mathcal{A}$ IS LOWER FREQUENCY THAN η

One important point of our method is that $\mathcal{A}$ varies on much bigger length scales then η. This length scale is determined by the Gaussian width s in the Gabor transform of η, Equation 5. Rewriting the Gabor transform as a convolution of a function with Gaussian shows that $\mathcal{A}$ indeed varies on the length scales s, instead of a smaller length scale determined by η.

First, express $\mathcal{A}$ by combining Equation 5 and $\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) = \zeta(\boldsymbol{x}, \boldsymbol{k}, t) e^{i\omega(k)t}$:

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) = \frac{1}{(2\pi)^2} \int_{\mathbb{R}^2} \eta_c(\boldsymbol{y}, t) e^{-\frac{|\boldsymbol{y}-\boldsymbol{x}|^2}{2s^2}} e^{-i(\boldsymbol{k}\cdot\boldsymbol{y} + \omega(k)t)} \, d\boldsymbol{y}. \tag{30}$$

Observe that the above integral is nothing but convolution between a function and Gaussian of width s:

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{k}, t) = \frac{1}{(2\pi)^2} \left(\eta_c(\boldsymbol{y}, t) e^{-i(\boldsymbol{k}\cdot\boldsymbol{y} + \omega(k)t)} *_{\boldsymbol{y}} e^{-\frac{|\boldsymbol{y}|^2}{2s^2}} \right)(\boldsymbol{x}) \tag{31}$$

Since the result of convolution is always as smooth as any of the two functions then the amplitude $\mathcal{A}$ has to be at least as smooth as the Gaussian. If s is much larger than the length scales in η, then the frequencies in $\mathcal{A}$ are much lower than those of η.

C CONTINUUM LIMIT OF WATER WAVE PACKETS

Jeschke & Wojtan [2017] introduce "water wave packets" as a Lagrangian primitive for surface water wave simulation. Under this model, the water height η can be computed as the sum of multiple wave packets:

$$\eta(\boldsymbol{x}, t) = \sum_j \hat{a}_j \, \phi\left(\boldsymbol{x} - \boldsymbol{y}_j\right) \, e^{-i(\boldsymbol{k}_j \cdot \boldsymbol{x} - \omega(k_j)t)} \tag{32}$$

where each packet in the summation consists of a scalar amplitude-like weight $\hat{a}_j$, a Gaussian envelope function ϕ centered at the packet's position $\boldsymbol{y}_j$, and a wave with representative wavevector $\boldsymbol{k}_j$. If we consider an *infinite number* of wave packets, spanning all possible packet positions $\boldsymbol{y}$ and all possible representative wavevectors $\boldsymbol{k}$, then we can express this sum as the following integral:

$$\eta(\boldsymbol{x}, t) = \int_{\mathbb{R}^2} \int_{\mathbb{R}^2} \hat{a}\left(\boldsymbol{y}, \boldsymbol{k}, t\right) \phi\left(\boldsymbol{x} - \boldsymbol{y}\right) e^{-i(\boldsymbol{k} \cdot \boldsymbol{x} - \omega(k)t)} \, d\boldsymbol{y} \, d\boldsymbol{k}. \tag{33}$$

We then note that the integral over $\boldsymbol{y}$ is just a convolution:

$$\eta(\boldsymbol{x}, t) = \int_{\mathbb{R}^2} \left(\int_{\mathbb{R}^2} \hat{a}\left(\boldsymbol{y}, \boldsymbol{k}, t\right) \phi\left(\boldsymbol{x} - \boldsymbol{y}\right) \, d\boldsymbol{y} \right) e^{-i(\boldsymbol{k} \cdot \boldsymbol{x} - \omega(k)t)} \, d\boldsymbol{k} \tag{34}$$

$$= \int_{\mathbb{R}^2} \left(\hat{a} * \phi\right)\left(\boldsymbol{x}, \boldsymbol{k}, t\right) e^{-i(\boldsymbol{k} \cdot \boldsymbol{x} - \omega(k)t)} \, d\boldsymbol{k}. \tag{35}$$

Now if we assume that $(\hat{a} * \phi) = \mathcal{A}$, then we arrive exactly at our Equation 7:

$$\eta(\boldsymbol{x}, t) = \int_{\mathbb{R}^2} \mathcal{A}\left(\boldsymbol{x}, \boldsymbol{k}, t\right) e^{-i(\boldsymbol{k} \cdot \boldsymbol{x} - \omega(k)t)} \, d\boldsymbol{k}. \tag{36}$$