Water Wave Animation via Wavefront Parameter Interpolation

STEFAN JESCHKE and CHRIS WOJTAN
IST Austria

We present an efficient wavefront tracking algorithm for animating bodies of water that interact with their environment. Our contributions include: a novel wavefront tracking technique that enables dispersion, refraction, reflection, and diffraction in the same simulation; a unique multi-valued function interpolation method that enables our simulations to elegantly sidestep the Nyquist limit; a dispersion approximation for efficiently amplifying the number of simulated waves by several orders of magnitude; and additional extensions that allow for time-dependent effects and interactive artistic editing of the resulting animation. Our contributions combine to give us multitudes more wave details than similar algorithms, while maintaining high frame rates and allowing close camera zooms.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Physically based modeling*

Additional Key Words and Phrases: ocean simulation, wavefront tracking, liquid animation, computational fluid dynamics

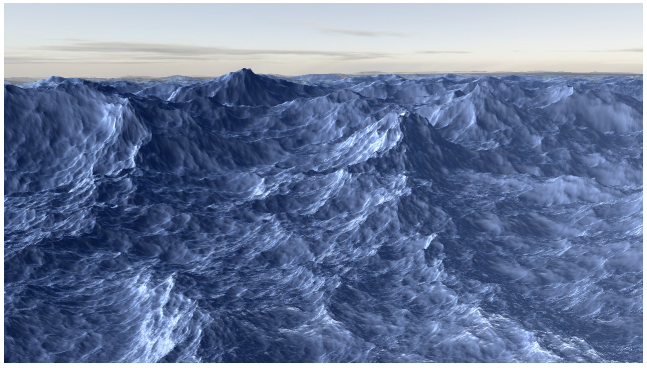


Fig. 1: In the simplest case of constant amplitude and velocity, our method exactly reduces to familiar spectrum-based wave simulation methods. While our approach does not have the speed of a fast Fourier transform, our dispersion approximation (§8.1) efficiently amplifies the amount of wave detail beyond previous approaches.

1. INTRODUCTION

Liquid simulation has been an important problem in computer graphics for decades. While fully three-dimensional simulations of the Navier-Stokes equations are essential for animating realistic splashes and breaking waves, procedural models are still the state of the art for animating large bodies of open water and creating highly detailed ocean textures. One of the primary reasons for this practice is that procedural Fourier spectrum-based deep water models [Mastin et al. 1987; Tessendorf 2004b] are far more efficient than 3D simulations and can produce high-frequency details without any Courant-Friedrichs-Lewy (CFL) limit.

Unfortunately, spectrum-based models rely on assumptions that only apply in a handful of situations, so they are conspicuously unrealistic in many important scenarios. Among other limitations, the spectrum-based models used in computer graphics are incapable of interacting with their environment through reflection, diffraction, and refraction. Consequently, the common practice is to approximate the correct behavior by compositing together spectrum-based methods with 2D or 3D Eulerian techniques. Previous research has proposed procedural models that account for some of these correct wave interactions, but they sacrifice the efficient computation and nearly limitless wave detail of spectrum-based techniques.

We present a new method capable of simulating waves that refract, reflect, diffract, and disperse while simultaneously exhibiting arbitrarily high resolution spatial details. Our algorithm is efficient, and several of our animations run at interactive or near-interactive frame rates. We categorize our method as a novel compromise between Eulerian and spectrum-based models; it combines general environment interactions with highly detailed surfaces, and it is free from time step restrictions and numerical damping.

Contributions

We first list two primary contributions which enhance the detail of existing ocean animation techniques. We then list three secondary contributions which offer additional improvements over the state of the art.

Multi-valued function representation (§6, §7) Our first major contribution is a novel technique that permits the reconstruction of high-frequency wave functions from their low-frequency phase and amplitude at arbitrary locations. This reformulation amounts to a change of variables that is practically independent from the Nyquist limit, and we take advantage of this fact with high-order interpolation and a sparse mesh data structure.

Dispersion approximation (§8.1) Our second major contribution is a simplification that shares data between similar waves. The approximation is exact in the common cases, converges under re-

This research was partially funded by the Austrian Science Fund (FWF) project P 24352-N23 and European Research Council (ERC) Starting grant 638176. The authors may be contacted at jeschke@ist.ac.at and wojtan@ist.ac.at.

DOI
http://doi.acm.org/

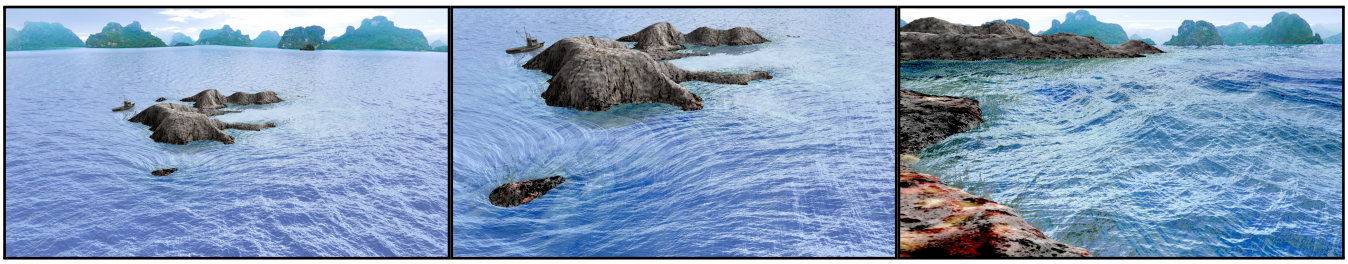


Fig. 2: Our algorithm efficiently creates water waves that reflect, refract, and diffract around objects. This figure shows how wave details are preserved at both large and small scales.

finement, and effectively amplifies the amount of visible wave detail with minimal computational overhead.

Wavefront behaviors (§5) Our wavefront tracking framework integrates dispersion, refraction, reflection, and smooth object diffraction. While these effects are commonplace in Eulerian fluid simulations, no other procedural wave method in computer graphics exhibits all of these effects simultaneously.

Time-dependence (§8.2) We introduce an amplitude blending function and a physically-correct viscosity model to support ripples and capillary waves.

Artist interface (§8.3) We describe an editing framework for interactively fine-tuning a liquid surface animation.

Existing spectrum-based ocean animation methods in computer graphics [Mastin et al. 1987; Tessendorf 2004b] animate rich water surfaces and represent the state of the art in fluid simulation detail. However, in practice these approaches are limited to periodic domains, cannot interact with boundaries, and are restricted to deep water scenarios. This paper generalizes spectrum-based methods to handle varying water depth and physically correct interactions with static obstacles, while preserving high levels of wave detail.

2. BACKGROUND

Before explaining our approach or the previous work in this field, we will first provide the reader with the relevant background for modeling simple surface water waves. All of the concepts in this section are results from linearized water wave theory and are explained in more detail in fluid dynamics textbooks such as [Lamb 1895; Dean and Dalrymple 1991].

2.1 Linearized water waves

Airy wave theory [Airy 1841] is a linear model that approximates the motion of surface waves on a body of water with a sum of sinusoidal functions:

$$\eta(\vec{\mathbf{x}}, t) = \eta_0 + \sum_{i=1}^{N} a_i \sin(\omega_i \cdot (\phi_i(\vec{\mathbf{x}}) - t)) \tag{1}$$

where η is the water height, η_0 is an arbitrary constant offset, a_i is the amplitude of wave i, ω_i is the angular frequency, t is the current time, and ϕ_i is a carefully chosen *phase function*.

Dispersion. Each wave has an angular frequency ω, which describes how quickly it oscillates over time. For surface water waves, the angular frequency is given by the *dispersion relation*:

$$\omega = \sqrt{\left(gk + \frac{\sigma}{\rho}k^3\right)\tanh(kh)} \tag{2}$$

where g is gravity, σ is the surface tension, ρ is the water density, h is the water depth at a given location, and k is the wavenumber. The wavenumber is inversely related to the wavelength λ by the relation $k{=}2\pi/\lambda$. Each wave propagates through space at a rate given by the *phase speed* c:

$$c = \frac{\omega}{k} = \sqrt{\left(\frac{g}{k} + \frac{\sigma}{\rho}k\right)\tanh(kh)} \tag{3}$$

Equation 3 describes many distinctive qualitative properties of water waves. Because $\tanh(\cdot){\approx}1$ for large inputs, waves with large h (deep water waves) or large k (capillary waves) travel at a speed almost independent of water depth; otherwise, the waves will speed up or slow down as the water depth changes. The $(g/k + \sigma k/\rho)$ term tends to infinity both as $k{\to}0$ and as $k{\to}\infty$, with a global minimum at $k{=}\sqrt{g\rho/\sigma}$. This means that both large and small wavelengths will travel quickly; in deep water, the slowest water waves are in between the two extremes at $\lambda{\approx}1.7$cm.

Phase function. The function ϕ in Equation 1 is chosen such that the resulting waves respect the initial conditions and propagate with phase speed c. This behavior is described by the *eikonal equation*:

$$|\nabla\phi| = \frac{1}{c} \tag{4}$$

The gradient of ϕ indicates the wave travel direction, and the level set $\phi{=}t$ indicates the location of the wavefront at time t. ϕ is also called the *travel time*, because the difference in ϕ between two points along a wave's travel path is precisely the time it takes for the wave to travel between them.

This method of choosing ϕ using the eikonal equation will yield exact solutions to the Airy wave model for large wavenumbers, but the error increases with the wavelength. This approach is commonly referred to as the "high-frequency approximation" in geometric optics, and it is considered accurate while the wavelength is smaller than the scales of boundary features and spatial variations in phase speed [Runborg 2007].

Conservation laws. The energy density of a given wave is

$$D = \frac{1}{2}\left(\rho g + \sigma k^2\right)a^2 \tag{5}$$

Note that D depends on amplitude. The energy per unit crest length

$$E = \int D dl \tag{6}$$

is equal to the energy density integrated over the length of the wavefront. Energy propagates across the water surface at a rate given by

the *group speed*:

$$c_g = \frac{d\omega}{dk} \tag{7}$$

which is generally different from any individual wave's phase speed. The *energy flux* $c_g E$ describes how energy is transported, and it is conserved by surface water waves:

$$\frac{d}{ds}(c_g E) = 0 \tag{8}$$

where s is a parameter along the direction of wave travel. In regions where c_g is constant, this equation reduces to the conservation of energy; amplitude decreases where wavefronts expand and increases where they focus. Equation 8 more generally describes the interplay between a and k, particularly *wave shoaling*: as waves enter shallow water and slow down, the wavelength shortens and the amplitude increases. In nature, waves may eventually become unstable and tumble over themselves, dissipating energy in the process. However, this wave breaking behavior is a non-linear phenomenon which is not captured by the linear Airy wave model.

Wave behaviors. Water waves exhibit the following common wave behaviors as they propagate through space:

Refraction: Different points on the wavefront may travel at different phase speeds, causing the wave to bend.

Dispersion: Waves with different wavenumbers will travel at different speeds, following Equation 3.

Reflection: When a wavefront hits an obstacle, it will bounce off. The incoming and outgoing angles are equal to each other.

Diffraction: When a wave grazes an obstacle, it bends around it.

2.2 Wavefront tracking

The main difficulty with using this theory to compute wave motion is the numerical solution of the eikonal equation (Equation 4) for ϕ. The extensive survey by Runborg [2007] reviews many methods used to solve this popular problem. Common techniques such as finite difference methods and the fast marching method [Sethian 1999] can efficiently produce a viscosity solution to the problem, but the most promising approach for our purposes is *wavefront tracking*.

The idea behind wavefront tracking is intuitive. We begin with a piecewise-linear curve representing a wavefront, and then we propagate the curve through space by updating the position of each vertex over time. The vertex locations are computed by integrating an ordinary differential equation, in a process equivalent to ray tracing.

By tracing the path of the wavefront as it evolves and noting the time it took for the front to arrive at each location, we can easily compute the travel time ϕ for each point in space. One of the strengths of this technique is that it permits wavefronts to fold over or intersect themselves, as they would in reality. In practice this means that $\phi(\vec{\mathbf{x}})$ is a *multi-valued* function, because a wavefront can cross a particular point in space any number of times. Once ϕ is computed and stored, it can be interpolated at any point in space and used to evaluate the wave height in Equation 1.

We stress that this multi-valued nature of the ϕ function is *not* a numerical artifact or implementation choice. Spatial projections of characteristic curves can intersect in general partial differential equations, and multi-valued phase is a fundamental property of the wave equation. Viscosity solutions to the eikonal equation (such as the fast marching method) assume that ϕ is a single-valued function and only compute the first crossing time. As a result, such methods prevent wave superposition, caustics, and reflections, and thus are not as flexible as wavefront tracking.

When ω is large, Equation 1 produces η with very high frequencies, even though the parameters ϕ and a only exhibit low-frequency behavior. (See Figure 3 for an illustration.) Wavefront tracking avoids dealing with η directly, so it can theoretically eliminate the high resolution grids required by Eulerian methods. However, until now, no previous method has actually achieved this in practice, as discussed in §3.3.

3. RELATED WORK

3.1 Ocean animation literature

One of the most convenient ways to solve Equation 1 is to assume that the phase function ϕ (Equation 4) is a linear function of position, thus reducing the solution to a closed form sum of constant-velocity planar waves. This is the approach taken by "spectrum-based" techniques common in computer graphics, and it can be further sped up by fast summation methods [Mastin et al. 1987; Tessendorf 2004b], and level-of-detail techniques [Hinsinger et al. 2002].

The main limitation of the constant velocity assumption is that it prevents general interactions between waves and their environment, such as refraction, reflection, and diffraction. In addition, it can only produce periodic waves that tile the fluid domain. Nevertheless, the efficiency of this technique enables the most detailed liquid simulations in computer graphics. The resulting waves are often modified to include spray, foam, and complex wave shapes [Fournier and Reeves 1986; Peachey 1986; O'Brien and Hodgins 1995; Gonzato and Le Saëc 1997; Thuerey et al. 2007; Thuerey et al. 2007]. Others use waves as boundary conditions, guide shapes [Nielsen and Bridson 2011; SideFX 2013], or texture maps to approximate additional details in more general fluid simulations [Chentanez and Müller 2010]. Researchers have also customized waves with artistic data [Nielsen et al. 2013]. These extensions are important for increasing visual realism and can easily be used with our model as well.

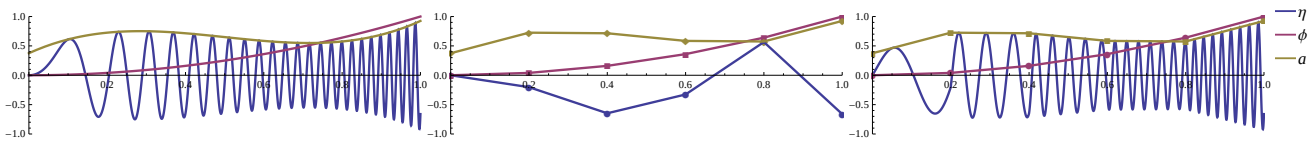
The works of Fournier and Reeves [1986] and Peachy [1986] simulate more general wave behavior by allowing the phase speed to vary. They numerically approximate Equation 4 by accumulating phase changes along straight lines in a regular grid. This approach is analogous to a fast marching solution of the eikonal equation; it cannot handle multi-valued ϕ functions, reflections, or large changes in direction. To remove such constraints, several researchers turn to wavefront tracking [Ts'o and Barsky 1987; Gonzato and Le Saëc 1997; 2000; Gamito and Musgrave 2002], which will be discussed in §3.3.

Yuksel et al. [2007] represents each wave crest with its own particle, allowing wave reflections and interaction with dynamic objects. However, it does not consider refraction or diffraction, and it cannot efficiently represent wave trains or high spatial frequencies. Keeler and Bridson [2014] introduce a boundary integral framework for animating ocean waves. This method simulates deep water waves interacting with obstacles, but the boundary integral is relatively expensive to evaluate and visible detail is limited by the resolution of the computational mesh.

The survey by Darles et al. [2011] provides a more thorough discussion of ocean animation techniques.

3.2 Eulerian fluid simulation literature

Simulating the full Navier-Stokes equations [Foster and Fedkiw 2001] and reductions like the shallow water equations [Kass and



(a) Ground truth (b) Standard approach – Interpolating η (c) Our approach – Interpolating ϕ and a

Fig. 3: Even though the wave parameters ϕ and a change slowly, the resulting wave function η can have very high frequencies (a). Interpolating discrete samples of η leads to aliasing (b), while interpolating ϕ and a first and then reconstructing η avoids this problem (c). This example uses linear interpolation for illustrative purposes, while our algorithm uses smooth interpolation for higher quality results (§7).

Miller 1990] and *iWave* [Tessendorf 2004a] add even more versatility to liquid animations. Solutions to the full fluid equations will of course demonstrate the desired wave behaviors of refraction, dispersion, reflection, and diffraction (§2.1). The shallow water assumption ($\tanh(kh) \rightarrow kh$) does not exhibit dispersion of surface gravity waves, but it allows reflection, refraction, diffraction, and interesting rotational motions. Like our approach, models based on linearized wave dynamics [Tessendorf 2004a] assume an irrotational velocity field, but they allow all four of the desired wave behaviors listed in §2.1.

While Eulerian discretizations exhibit many desirable properties that simple procedural models lack, they also have fundamental limitations. Most importantly, Eulerian approaches are bound by Nyquist's theorem: the simulation resolution limits the maximum wave detail. Adaptive techniques help by increasing resolution [Losasso et al. 2004; Ando et al. 2013], but they cannot compete with methods specifically designed to capture high-frequency waves. Eulerian models are also prone to time step limitations (the CFL condition) which become more restrictive as spatial resolution increases. They also require complex treatment of non-reflective open water boundary conditions [Söderström et al. 2010], which are trivial for Lagrangian methods like wavefront tracking. Lastly, Eulerian discretizations often require artificial damping for stability, or they exhibit numerical dissipation from implicit schemes or iterated re-sampling operations. This lack of energy conservation makes it surprisingly difficult for many Eulerian models to simulate waves traveling over long distances. We weigh the benefits and drawbacks of our method against those of a popular Eulerian method in §10.

3.3 Wavefront tracking literature

Previous wavefront tracking algorithms in computer graphics and computational physics employ one of two methods:

(1) Store the entire wavefront at all time steps, recording the wave parameters ϕ and a at each vertex. Connect each wavefront vertex to its neighbors in space and time via quadrangulation. Use this wavefront mesh directly for rendering by evaluating each individual wave height η_i at the mesh vertices, linearly interpolating η_i across quads, and summing up η_i wherever quads overlap [Gonzato and Le Saëc 2000].

(2) Create a regular grid that covers the spatial domain. When a wavefront segment crosses a grid point, interpolate wave data ϕ and a onto it. After wavefront propagation is complete, each grid point might hold the data of several different waves. Evaluate the total wave height η at each grid point, and interpolate η at render time [Ts'o and Barsky 1987; Bulant and Klimeš 1999; Gamito and Musgrave 2002].

Both of these options animate accurate and detailed wave trains, but they also have drawbacks. They each sample and interpolate the high frequency η function, so small wavelengths are impossible to represent without extremely dense sampling (Figure 3). Neither of these data structures are spatially adaptive, so they require significant computation and memory even for mundane animations (Figure 4), and the memory consumption additionally scales with the total number of waves in the animation. Thus these methods are practically limited to a small handful of wave trains, instead of the hundreds of waves desired for a realistic animation. These shortcomings are often obscured with unphysical noise textures [Gonzato and Le Saëc 2000; Gamito and Musgrave 2002].

Various wave behaviors are achieved by modifying the trajectory of wavefront vertices. Refraction is incorporated using Snell's law [Ts'o and Barsky 1987; Gonzato and Le Saëc 1997] or a more accurate geometric optics approach [Gamito and Musgrave 2002]. Dispersion is a default behavior in all methods that results from tracking more than a single wavefront, but tracking large numbers of waves is often inefficient for the reasons mentioned above.

Reflection and diffraction behaviors are surprisingly absent from most ocean animation techniques, with notable exceptions by Gonzato and Le Saëc. Their initial work [1997] does not claim to handle reflection or diffraction, but we show in §5.1 how a method similar to their "aground" waves can actually model diffraction around smooth objects. Their follow-up work [2000] explicitly models reflection and diffraction, but they chose to damp out reflection waves, and the diffraction model is only applicable to sharp corners. None of these methods simultaneously incorporate refraction, dispersion, reflection, and diffraction in an environment with general boundary shapes.

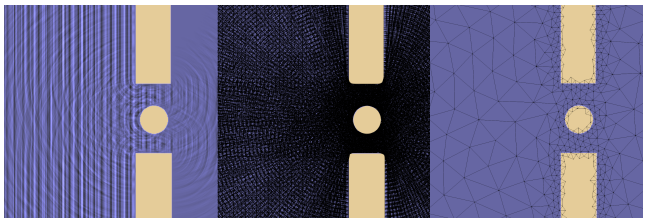


Fig. 4: Waves interact with obstacles and create interference patterns (left). Storing the entire wavefront at all time steps can be impractical (middle). Our method efficiently represents the same wavefront information with a coarse triangle mesh (right), which reduces memory usage by a factor of 17 in this example.

4. METHOD OVERVIEW

Our method builds upon previous wavefront tracking approaches to efficiently simulate several orders of magnitude more waves while evading the Nyquist limit on wavelength. In a precomputation step, we first track the wavefronts (§5) in a manner that respects all of the desired wave behaviors from §2.1, storing the wave parameters on the vertices of a low-resolution adaptive triangle mesh (§6.1). In a crucial preparatory routine, we also track how each set of sampled wave parameters matches up with its neighbors (§6.2), and we use this information to represent the multi-valued parameter functions continuously within each triangle (§6.3).

During runtime, we adaptively sub-sample the triangle mesh in a view-dependent manner. At each of these sub-sample locations, we smoothly interpolate the wave parameters and compute the wave displacements (§7). We also introduce a dispersion approximation that efficiently magnifies the number of simulated waves, a physics-based damping model to realistically model small-scale details, and an artistic editing framework for post-process fine-tuning (§8).

5. WAVEFRONT PROPAGATION

The basics of our wavefront propagation algorithm are the same as existing methods. We start with a piecewise linear curve representing our wavefront. Each wavefront vertex is assigned an initial wavenumber k_i, and each wavefront segment is assigned an initial energy per unit crest length $E{=}\int D dl{\approx}DL$, where L is the length of the segment.

Airy wave theory asserts that the angular frequency $\omega(k,h)$ is constant for each wavefront, so k must vary with water depth h. We introduce a fixed-point iteration scheme based on Equation 3 to compute k for this purpose: We first initialize k to the most recent wavenumber value, and then we iterate $k := \omega/c(k,h)$ until convergence. In our experience this scheme always converges, and it is reliably faster than Newton iteration. This new value of k is used for computing a more accurate energy density and phase speed.

During each time step, we evaluate c at each vertex using Equation 3, and we choose the propagation direction according to Snell's law [Ts'o and Barsky 1987]. The direction and speed give us a phase velocity vector $\vec{\mathbf{c}}$, and we advance each vertex with symplectic Euler integration. We could replace Snell's law and Euler integration with more accurate methods [Gamito and Musgrave 2002], but we found that the simpler choices produce acceptable results.

The energy density is updated according to Equation 8. We first use finite differences to approximate the group speed at wavefront vertices,

$$c_g(k,h) \approx \frac{\omega(k+\Delta k,h)-\omega(k,h)}{\Delta k} \tag{9}$$

with $\Delta k{=}10^{-6}$ (note that c_g can be evaluated analytically as well). We then interpolate c_g to the wavefront segments and scale the new energy density such that it satisfies $c_g^{\text{old}} D^{\text{old}} L^{\text{old}} = c_g^{\text{new}} D^{\text{new}} L^{\text{new}}$, the discrete version of Equation 8. Note that this simple equality constraint preserves the energy flux of each wavefront segment up to numerical precision; there are no mechanisms for numerical drift or artificial dissipation of energy. The amplitude is then computed by solving Equation 5:

$$a = \sqrt{\left(\frac{L^{\text{old}}}{L^{\text{new}}}\right)\left(\frac{c_g^{\text{old}}}{c_g^{\text{new}}}\right)\left(\frac{2D^{\text{old}}}{\rho g+\sigma k^2}\right)} \tag{10}$$

This scaling enforces the wave expansion, focusing, and shoaling effects discussed in §2.1. As noted earlier, steep waves in nature tend to dissipate energy through non-linear breaking, which is not captured here. The linear theory also allows *caustics* to produce unphysically large amplitudes when $L{\to}0$. Instead of switching to a more accurate but complex method, we approximate nonlinear dissipation by removing energy from a wavefront segment until a prescribed steepness threshold a/λ is met. We use 0.07 as the maximum steepness based on the theory of breaking waves in deep water [Dean and Dalrymple 1991].

We adaptively subdivide wavefront segments whenever their lengths L are beyond a maximum length or the angle between neighboring wavefront segments is too large, and we similarly collapse edges that are below a minimum length and have a small angle between neighboring segments. We conserve energy by evenly distributing the DL quantity from the original segment to the new ones during subdivision, and by summing up DL values when collapsing wavefront segments. We also delete any wavefront segments whose amplitude is below a minimum threshold, and the wavefront propagation ends when all segments are deleted or have left the simulation domain.

5.1 Desired wave behaviors

The first two in our list of desired wave behaviors are already accounted for: *refraction* happens whenever c changes based on h, and *dispersion* occurs naturally when we simulate multiple wavefronts with different wavenumbers. Interactions with boundaries (reflection and diffraction) require additional work.

The basic behavior of a reflecting wavefront vertex is identical to ray tracing [Whitted 1980]. The vertex intersects the boundary and then changes direction so that its angle of reflection about the boundary normal is equal to the angle of incidence. We also invert the amplitude of the reflected wavefront, based on the analytical solution of the wave equation near a reflecting boundary. If a wavefront vertex hits the boundary at a grazing angle (more than 82° with the boundary normal in our implementation), it will diffract rather than reflect. Instead of bouncing off the object, the diffracted vertex continues on a path tangent to the surface at speed c. We note that this behavior is similar to the "aground" ray behavior of [Gonzato and Le Saëc 1997], though they neither acknowledge it as diffraction nor simultaneously handle reflection.

When one vertex of a wavefront segment reflects within a time step and the other vertex does not, the two vertices are oriented in opposite directions, the amplitudes are of opposite signs, and interpolating along the wave segment makes little sense. In reality there would be a point somewhere between the two that lies exactly on the boundary surface. To avoid these interpolation problems, we find this boundary point with a bisection search along the segment, create a new vertex there, and connect the oppositely-oriented endpoints to this boundary vertex. In the next time step, this boundary vertex will either reflect (and probably soon be deleted with an edge-collapse operation) or diffract and begin stretching out the wavefront along the object boundary. This diffraction behavior is only guaranteed to work for smooth boundaries, but we handle sharp corners in practice by smoothing out the normal field along the surface (by blurring a level set representation, for example).

The distinctive bending appearance of diffracting waves is a side effect of the repeated stretching and subdivision of the wavefront as the diffracting vertex curves along the boundary shape. Conveniently, this behavior is in line with the geometric theory of diffraction [Levy and Keller 1959], which states that the amplitude at the original grazing point a_0 is related to the amplitude at a point fur-

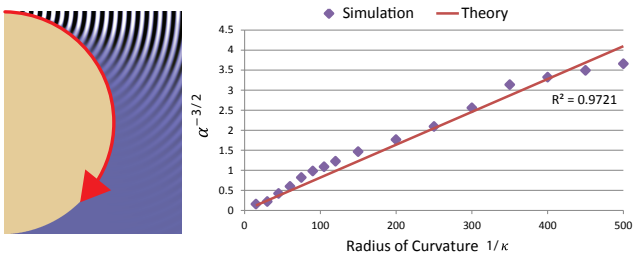


Fig. 5: We simulated diffraction around circular boundaries of varying radii, and the amplitudes decay exponentially along the path marked in red (left). Our measured decay rate matches the theoretical dependence on boundary curvature from Equation 11 (right).

ther along the boundary $a(s)$ by the relation:

$$a(s) = a_0 \left(\frac{d\sigma_0}{d\sigma} \right)^{\frac{1}{2}} \exp\left(- \int_0^s \alpha(r) dr \right) \qquad (11)$$

$$\text{with} \quad \alpha(r) \propto k^{1/3} \kappa^{2/3}$$

Here, $d\sigma_0$ represents the length of an infinitesimally small wavefront segment at the grazing point and $d\sigma$ is the total length of that segment after it has stretched out due to diffraction. The ratio $d\sigma_0/d\sigma$ basically characterizes the change in length of a wavefront segment, and this exact square-root scaling law is already built into our computation of amplitude in Equation 10 as a result of energy conservation.

The $\exp(\cdot)$ term on the right of Equation 11 states that the amplitude will exponentially decay at a rate related to the integrated geodesic curvature κ of the boundary along the path of diffraction. Intuitively, if the boundary has no curvature then there is no decay. Similarly, if the boundary has constant curvature, then the diffracting vertex will repeatedly pull away from its original trajectory and cause the amplitude to shrink. This behavior is also handled naturally by our method, because a diffracting vertex that repeatedly stretches the wavefront will repeatedly cause segments to subdivide. Each subdivision cuts the energy in half, so repeated subdivision leads to an exponential decay.

We test our results against the theoretical prediction in Figure 5, and we discuss this experiment in detail in Appendix A. Our diffraction method is currently erroneously independent of wavenumber k, and we leave this as an interesting direction for future work.

6. RECORDING WAVEFRONT PARAMETERS

The previous section explained how we propagate the wavefront across the liquid domain in a physically appropriate manner. As the wavefront translates and deforms, we wish to store its simulation variables for later use. For this purpose, we create a low-resolution spatially-adaptive triangle mesh (~5000 triangles for the examples in this paper) that covers our water domain. We add more mesh resolution near boundaries and slopes in the seabed, where the wavefront will tend to curve the most. To actually create the triangle meshes in our examples, we first generated a Poisson-disk sampled point set with the disk radius varying with boundary distance, and then we created a Delaunay triangulation of those points using the *Triangle* software package [Shewchuk 1996].

6.1 Recording ϕ and a at mesh vertices

During propagation, we detect when a wavefront segment crosses a triangle mesh vertex using continuous collision detection [Moore and Wilhelms 1988], which amounts to the solution of a quadratic equation. The endpoints of the wavefront segment during the time steps immediately before and after intersecting each mesh vertex form a quadrilateral, and we can use bilinear interpolation to transfer information from the four endpoints onto the mesh vertex. Because a is normally stored on wavefront segments, we temporarily average it onto the wavefront vertices for this interpolation process.

We choose to store the travel time ϕ and the amplitude a at a mesh vertex for each intersection, and we also calculate and store their derivatives as explained in Appendix B. One of the derivatives depends on phase velocity, so we store $\vec{\mathbf{c}}$ at each mesh vertex as well.

Because each wavefront can cross a single location multiple times, ϕ and a are *multi-valued* functions over space. When we store wavefront variables at mesh vertices, we are sampling from this multi-valued function: each vertex can have an entire list of different ϕ variables, one for each time the wavefront intersected that position. From this information, we can easily evaluate Equation 1 at the vertices if we wish, and this per-vertex evaluation is precisely what all previous wavefront tracking methods do. However, a naïve per-vertex evaluation preserves the multi-valued function property at vertices *only*. We require the ability to evaluate the multi-valued function within a *triangle* in order to avoid the Nyquist limit (Figure 3).

We introduce a method for interpolating multi-valued functions within a triangle using the following three-step process:

(1) Divide the multi-valued function into a set of continuous single-valued functions (one for each wavefront that crossed the triangle, if possible).
(2) Interpolate each single-valued function within the triangle.
(3) Treat each single-valued function as a separate wave in Equation 1.

The first step is trivial in the common case of a single wavefront translating through space, but it is far from straightforward in the presence of general reflections and refractions. For example, different wavefronts may intersect mesh vertices in different orders, a single wavefront may intersect the same mesh vertex multiple times during a sharp refraction, and wavefronts never intersect mesh vertices that lie within a reflecting boundary.

We overcome these complications by first solving the simpler problem of finding a single-valued function along a mesh edge between two mesh vertices. We then extend this idea to find a single-valued function that interpolates corresponding function samples across an entire triangle.

6.2 Single-valued functions along mesh edges

As stated above, a function sample (ϕ or a) is created when the wavefront crosses over a mesh vertex. This wavefront can then propagate from the original mesh vertex along one of its adjacent mesh edges, and then eventually cross the mesh vertex at the other end, creating a new function sample. Because we know that the wavefront traversed directly along a mesh edge to connect two vertices, we can safely assume that these two function samples are related and can be interpolated. This would not be the case, for example, if the two function samples were created by different wavefronts, or if the wavefront turned around and exited the triangle before re-entering sometime later. Our basic strategy here is to iden-

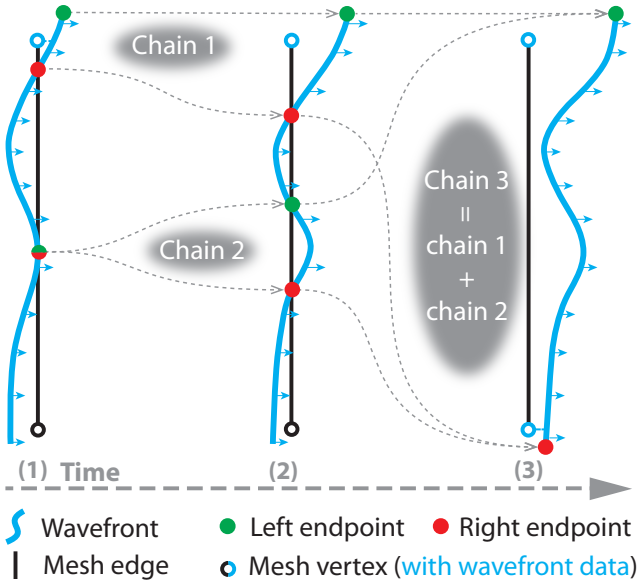


Fig. 6: A top-down view of our wave chain computation as the wavefront (blue) crosses a triangle edge (black). This example occurs over the course of three time steps (left, middle, right). For details please refer to the text.

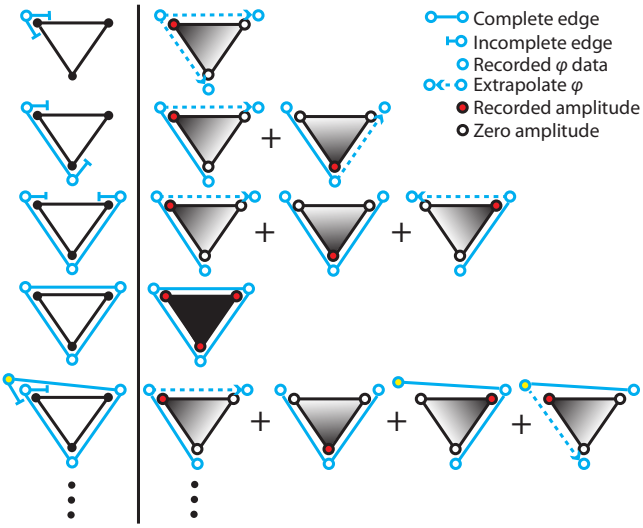


Fig. 7: Left: Different edge sets (§6.2) formed when a wavefront intersects a triangle. Right: Output single-valued functions over the triangle (§6.3) for each edge set. The fourth row is the only possible case without any incomplete edges; all other cases on the left column feature the wavefront as a single connected component bound by two incomplete edges. The left column can be straightforwardly extended to all possible scenarios by wrapping the wavefront around the triangle further.

tify related function samples along mesh edges by paying attention to the path of the propagated wavefront. In cases where this task becomes too complicated, we will extrapolate the function samples across edges in a way that avoids introducing artifacts.

A detailed wavefront may intersect a mesh edge at multiple points at a given time, and it may cross at variable speeds. To address these complications, we introduce the concept of a wave *chain*, which represents a continuous section of wavefront geometry that has already crossed over a specific mesh edge (see Figure 6). Each mesh edge can have a number of associated chains, and each chain is represented computationally by referencing the leftmost and rightmost wavefront vertices that have already crossed the mesh edge. For each mesh edge, we create a new chain whenever an isolated section of the wavefront crosses it, and we update these chains as they evolve and grow over time. When some straggling wavefront geometry finally crosses the middle of a mesh edge, the two chains that were previously separated will merge (by replacing them with a single new chain that spans the union of their wavefront geometry). When a chain segment crosses a mesh edge endpoint, we note exactly which ϕ and a samples were created (see the last paragraph) and store a reference to them in the chain. Figure 6 illustrates these ideas.

Once the chain has moved on and no longer intersects the edge, we can confidently interpolate the two endpoint function samples that were associated with the chain. For example, if two function samples ϕ_A and ϕ_B were associated with a wave chain when it finished intersecting with the mesh edge e, then we can interpolate ϕ_A and ϕ_B across e using linear or Hermite interpolation. We call this new association between function values a *complete edge*. Sometimes a wave chain can finish intersecting an edge and only have one associated function value, say ϕ_C. This behavior can occur, for example, in the event of a wavefront reflection or if wavefront geometry is deleted due to low energy. In this case, we note that ϕ_C has no other associated function sample along this edge, and we call this incomplete association an *incomplete edge*. If the finished chain has no function samples associated with it, then we simply discard it.

6.3 Single-valued functions within mesh triangles

Given the *complete* and *incomplete edges*, we wish to connect these function-associations together so that we can interpolate our function anywhere within a triangle. The basic principle is the same as when we connected function samples along mesh edges using wave chains: we link together *complete* and *incomplete edges* if they arose from the same wavefront. Specifically, if two pairs of edge associations share identical ϕ and a samples at their common vertex, we group them together. These grouped edges represent sets of interpolatable data for each triangle. Most of these edge sets form closed triangles (Figure 7, fourth row from the top) composed of three complete edges. These closed triangles only share compatible function samples with each other, and we can interpolate the function anywhere within the triangle using techniques described in §7.

All other sets of edges (in particular, sets ending with incomplete edges, see Figure 7) will not form simple closed triangles, so we treat them differently. We decompose the interpolation into several separate basis functions: one for each vertex with a valid function sample. Each basis function assigns the original ϕ and a data to the valid vertex (red vertices in Figure 7) and assigns default data to the other vertices: the default a data are a=0 and ∇a=0 (which smoothly fades the wave out of existence), and the default ϕ data are set to the recorded values if possible and extrapolated otherwise (if the vertex data is missing from an incomplete edge). We can then use the techniques in §7 to interpolate each basis function anywhere within the triangle.

Once we have constructed our basis functions for all triangles, it is possible to interpolate these wave parameters and use Equation 1 to treat each basis function as a separate wave. Because edge

data is treated identically in neighboring triangles and amplitudes smoothly fade to zero along incomplete edges, this scheme guarantees that the final wave heights will be continuous over all space. In addition, the default phase data are chosen to make ϕ consistent wherever possible, and the default amplitude data give our basis functions the compact support and delta function properties desirable in finite element analysis. These properties are important during the final wave height summation (Equation 1), because they allow the interpolation of function samples even in the difficult cases of rows 1, 2, 3, 5, etc. in Figure 7.

7. INTERPOLATING WAVE PARAMETERS

At this point in our algorithm, the precomputation phase has finished. We now have functions for ϕ and a constructed over a triangle, and we can interpolate them and compute η wherever we wish. A major benefit of this concept is that we can reconstruct high-frequency waves even with a coarse triangle mesh (Figure 1 shows an extreme example with only two triangles). However, we must be careful when interpolating high frequency waves across mesh edges, because they tend to highlight discontinuities between triangles. To eliminate this problem, we considered higher-order C^1 schemes. Our implementation uses the cubic side-vertex interpolation scheme [Nielson 1979], which needs first derivatives but does not require any information from neighboring triangles. For completeness, we supply the equations for the side-vertex method in Appendix C.

We found that interpolating ϕ^2 and then taking its square root produces more realistic wave motions than interpolating ϕ itself, as noted by Ursin [1982]. We give details in Appendix B, and we compare different interpolation schemes in Figure 9.

7.1 Reconstruction

Once we know how to interpolate a and ϕ, we can reconstruct η at a given point in space. We implement a level of detail approach similar to Hinsinger et al. [2002] to decide which points are evaluated: we first use GPU acceleration to create a pixel grid onto the viewport and then project the pixel locations onto the coarse planar triangle mesh representing the water domain. At each of these sample points, we use GPU acceleration to interpolate a and ϕ and evaluate η using Equation 1. For additional non-linear effects, we use the Biesel wave model described in [Fournier and Reeves 1986] (a generalization of the deep water Gerstner wave model) to displace the surface in both vertical and horizontal directions, instead of only using η for height field displacements.

8. EXTENSIONS

8.1 Dispersion approximation

To display realistic dispersion behavior in our simulations, we should animate hundreds of waves with slightly differing wavenumbers. We can certainly do this already by propagating a separate wavefront for each k_i and then storing their ϕ_i and a_i functions, as described earlier. However, storing and interpolating all of these variables will quickly become a computational burden. We can conveniently remove this burden and speed up computation by several orders of magnitude if we assume that wavefronts with similar wavenumbers have similar ϕ_i and a_i functions. We make the approximation that, for two waves with identical initial conditions but slightly different wavenumbers, one wave's ϕ and a functions are a constant multiple of the other's. This "dispersion approximation" is exact for common cases like deep water, shallow water, capillary waves, and flat sea beds, and the error converges to zero with refinement. Appendix D analyzes the validity and errors in more detail, and Figure 8 illustrates the visual differences.

We implement our dispersion approximation as a generalization of Equation 1. Each wave i re-uses its tracked information to calculate several similar waves ij:

$$\eta(\vec{\mathbf{x}}, t) = \eta_0 + \sum_{i=1}^{N} \sum_{j=1}^{M} a_{ij} \sin(\omega_{ij} \cdot (\phi_{ij}(\vec{\mathbf{x}}) - t)) \qquad (12)$$

where M is the number of approximate waves that we will associate with wave i. The amplitude, wavenumber, frequency, and phase of the new waves are represented by a_{ij}, k_{ij}, ω_{ij}, and ϕ_{ij}. Our dispersion approximation assumes $\phi_{ij}{=}C_{ij}\phi_i$ and $a_{ij}{=}A_{ij}a_i$ to get

$$\eta(\vec{\mathbf{x}}, t) = \eta_0 + \sum_{i=1}^{N} \sum_{j=1}^{M} A_{ij} a_i \sin(\omega_{ij} \cdot (C_{ij}\phi_i(\vec{\mathbf{x}}) - t)) \qquad (13)$$

where $A_{ij}{=}a_{ij}/a_i$ is a constant ratio of amplitudes, and $C_{ij}{=}c(k_i)/c(k_{ij})$ is the constant ratio of initial phase speeds, as discussed in Appendix D. Setting $A_{ij}{=}1$ and $k_{ij}{=}k_i$ reproduces the original wave i.

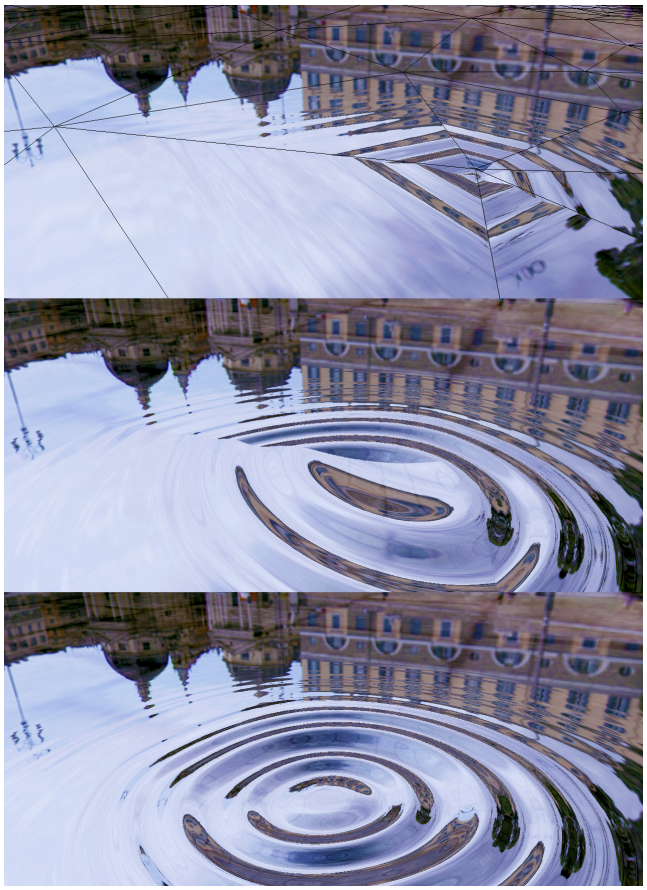
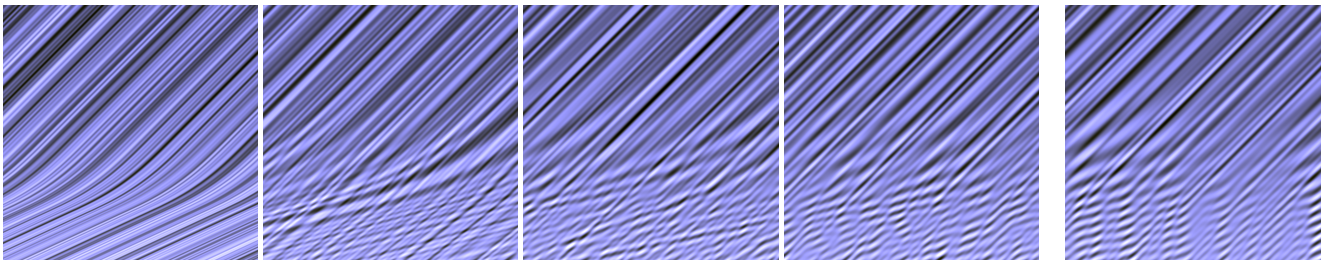


Fig. 9: The choice of a C^1 interpolant greatly affects the smoothness of the reconstructed waves. Here we illustrate (from top to bottom) linear, cubic Hermite, and side-vertex interpolation schemes for interpolating ϕ^2. The same mesh and phase function is used in all three examples.



(a) 1 wavefront (b) 2 wavefronts (c) 4 wavefronts (d) 8 wavefronts (e) Ground truth

Fig. 8: This scenario causes waves to refract in a wavenumber-dependent manner toward the bottom of each image. Instead of simulating all 24 wavefront travel paths (e), our dispersion approximation simulates multiple waves along fewer independent travel paths. Images (a)–(d) show how the interference pattern converges as we simulate more wavefronts.

8.2 Time-dependent amplitudes

The algorithm so far considers wave amplitudes as a function of space, but not time. Let us say that some wavefront i represents a set of waves rippling away from some initial disturbance in the water surface. We assume that the disturbance occurs at time τ_i, and it emits several dispersion waves ij. For some global time t, the quantity $t-\phi_i(\vec{\mathbf{x}})-\tau_i$ is negative before the wavefront crosses point $\vec{\mathbf{x}}$ and positive afterward. We modify this idea to accommodate all dispersion waves and define the time since wavefront arrival $T_{ij} \equiv (t - C_{ij}\phi_i - \tau_i)$.

Our method can simulate splash waves (waves in an initially calm area that suddenly race outward from a disturbance) by multiplying a_{ij} by a time-dependent blending function $B(T_{ij})$, which is equal to zero while $T_{ij}<0$ and then blends quickly up to 1 afterward. We cease the emission of splash waves by driving $B(T_{ij})$ back to zero after the wave source is removed. The decay rate can in principle be chosen in order to conserve energy, but we leave it as a user parameter. A rule of thumb for setting the splash wave amplitudes and wavenumbers is that they should approximate the Fourier transform of the shape and velocity profile of the initial disturbance; a sinking ship creates larger wavelengths than a raindrop.

We also extend our model to allow viscous damping. Airy wave theory is based on inviscid potential flow and exhibits no damping, but Padrino et al. [2007] shows that re-deriving the theory based on *viscous* potential flow produces similar wave motion with an amplitude equal to

$$a_{ij}^{\text{visc}} = a_{ij} e^{-\nu k_{ij}^2 C_{ij} \phi_i} \tag{14}$$

where a_{ij} is the undamped amplitude and ν is the kinematic viscosity of water. Note that waves with large wavenumbers (capillary waves in particular) decay quickly, while waves with small wavenumbers are essentially undamped. Our accompanying video shows some of these time-dependent amplitude effects.

8.3 Wave editing

An artist may prefer to interpret the results of our wavefront propagation simply as a suggestion, and then fine-tune them in an interactive editing session. We implemented an interactive painting interface which uses various brushes to increase or decrease the amplitude functions of each wave. We also implemented filters to isolate groups of waves traveling in a specific direction or exhibiting a specific range of wavelengths (Figure 10).

While it is straightforward to locally adjust a and globally rescale ϕ and k of each wave, we do not recommend locally modifying ϕ. The wave speed is related to the *gradient* of ϕ (Equation 4), so local changes to ϕ may influence the motion in counterintuitive ways. In particular, adding critical points to ϕ will force waves to travel backwards. It may be possible to solve this problem with a gradient-domain painting technique [McCann and Pollard 2008] extended to multi-valued functions.

We can also change the wave function profile by locally adjusting Biesel or Gerstner wave parameters or blending together artist-defined wave functions in place of the $\sin(\cdot)$ function in Equations 1 and 13. We store all manual changes to parameters onto our coarse triangle mesh, but we could allow even finer control by creating a separate, high-resolution parameter adjustment map.

9. ALGORITHM SUMMARY

To review, the algorithm first propagates a discretized wavefront curve through a 2D domain, as outlined in Algorithm 1. During this propagation phase, we advance the wavefront with numerical integration (§5), store wave parameters at the vertices of a coarse triangle mesh (§6.1), and compute the connectivity information necessary to interpolate functions across each mesh triangle (§6.3).

During runtime, the level-of-detail (LOD) system selects dense sampling points in a view-dependent manner (§7.1), and we interpolate ϕ and a at these locations (§7) and evaluate wave heights. We also evaluate additional wave heights using our dispersion approximation (§8.1) and optionally add time-dependence to their amplitudes (§8.2). We are then free to add additional effects (we displace each wave horizontally using the Biesel model, as in [Fournier and

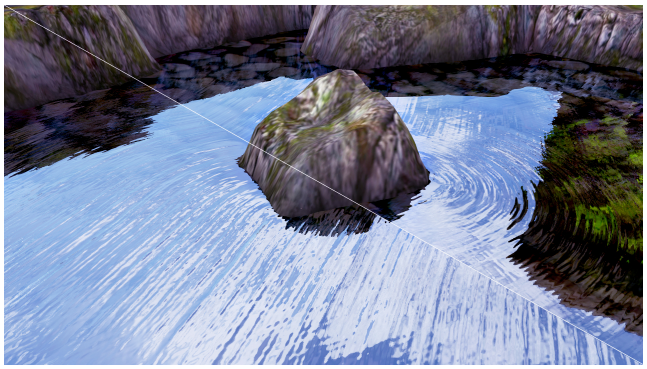


Fig. 10: Here we use our painting interface to fine-tune an existing wave simulation. The upper right half of the image amplifies waves reflecting off the stone, while the bottom left image emphasizes waves traveling in the opposite direction. Please see our supplemental video for a more thorough demonstration of the editing interface.

Algorithm 1: Precomputation

input : Set of polylines representing wavefronts
Coarse planar triangle mesh $\mathcal{M}$ covering fluid domain
output: Interpolatable set of ϕ and a samples stored on $\mathcal{M}$

foreach *wavefront* W_i **do**
  $\phi_i := 0$;
  while W_i *exists* **do**
    Advance W_i using numerical integration; // §5
    $\phi_i \mathrel{+}= \Delta t$;
    Update a and other propagation variables;
    if W_i *crosses vertex* v *in* $\mathcal{M}$ *during this timestep* **then**
      Store ϕ_i, a_i, and their derivatives on v; // §6.1
    if W_i *crosses edge* e *in* $\mathcal{M}$ *during this timestep* **then**
      Update edge chains associated with e; // §6.2
    if *edge chain* β *finished during this timestep* **then**
      Add β to edge sets; // §6.2
      Create new triangle basis functions for ϕ and a where appropriate; // §6.3
    Subdivide, merge, or delete appropriate W_i segments;

Algorithm 2: Runtime

input : Interpolatable set of ϕ and a samples stored on triangle mesh $\mathcal{M}$
output: Animation of detailed water waves

while *Animating* **do**
  Create a level-of-detail grid $\mathcal{G}$ on the screen;
  Project $\mathcal{G}$ onto $\mathcal{M}$;
  foreach *Triangle* T *in* $\mathcal{M}$ **do**
    foreach *grid point* p *in* $\mathcal{G}$ *inside* T **do**
      $\eta := 0$;
      foreach *set of* ϕ_i *and* a_i *functions in* T **do**
        Interpolate ϕ_i and a_i at p;
        foreach *dispersion approximation wave* W_{ij} **do**
          Evaluate the height η_{ij} of wave W_{ij};
          $\eta \mathrel{+}= \eta_{ij}$;
  Add post-process effects like spray, foam, or displacement;
  Render the scene;

Reeves 1986]) and then render the scene. All of these steps are executed in parallel on the GPU. The runtime algorithm is outlined in Algorithm 2.

10. RESULTS

Boundless ocean. In deep water, our method reduces to a summation of periodic sine waves with constant amplitude and phase speed. This behavior is identical to spectrum-based methods and specifically reduces to that of [Hinsinger et al. 2002]. Our method does not benefit from FFT optimization, but it is trivially parallelized and ideal for GPU implementation. Figure 1 illustrates this idea by animating a detailed ocean with a mesh made of only two giant triangles. The example was produced by propagating 36 wavefronts (sampling a circle in 10° increments) with 400 dispersion-approximation waves each, for a total of 14,400 waves.

Island. Figure 2 illustrates many desirable wave behaviors, including refraction over a sandbar and the sloping sea floor, diffraction as the waves curve around the island, reflection as waves bounce off the rocky coast, and dispersion as many waves travel with different speeds. This single scenario illustrates several interesting wave regimes: deep water behavior occurs far from the island while shallow water behavior occurs closer to the coast; the sea is noisy in the open ocean while it is peaceful in a shadowed bay; regular wave patterns occur near simple geometry, while more complicated geometry and changing sea depth cause chaotic wave interference. Our approach allows for a seamless transition between all of these distinct behaviors.

Pond & puddle. Our accompanying video shows a large disturbance similar to a rock plunging into a pond and a small splash similar to a raindrop in a puddle. Both examples exhibit time-dependent amplitudes (§8.2), but the effects vary naturally depending on scale. The pond splash has little damping, and the long wavelengths outrun the shorter ones. The puddle splash has stronger damping, and it exhibits *anomalous dispersion* in which surface tension causes the shorter wavelengths to outrun the larger ones (Figure 9). These effects are exactly in line with theoretical predictions.

Wave editing. We can modify the output of our algorithm using a simple painting interface (Figure 10). The user increases or decreases wave amplitudes by interactively painting onto the water with brushes of various shapes and sizes, specifically targeting waves traveling in different directions with a velocity filter.

10.1 Performance

The cost of our precomputation (Algorithm 1) scales with the number of wavefront segments that are propagated, multiplied by the number of simulation time steps that each segment exists. The costs of recording ϕ and a samples and creating triangle basis functions (§6) is negligible in comparison. We implemented basic adaptive time-stepping to speed up these computations, but many further strategies could be implemented if more efficiency is desired. In particular, the precomputation can be perfectly parallelized by tracking each wavefront independently and combining the results afterward. Precomputation for the Island scene was trivially sped up from 30 minutes to 9 minutes by tracking wavefronts in parallel across 6 processors. With a slightly more aggressive adaptive time stepping strategy, we were able to further reduce this time down to 5 minutes with visually indistinguishable results. A more principled or asynchronous adaptive approach could conceivably reduce our precomputation time further. More practically, we can generate a quick preview by pre-computing only a few representative wavefronts at first and then adding more detail later if necessary.

The runtime performance (Algorithm 2) is dominated by the total number of wave evaluations, $O(HNM)$. Here, H is the number of evaluated wave heights based on the number of points in the level-of-detail grid, N is the number of propagated wavefronts, and M is the number of additional dispersion approximation waves. The number of interpolations is $O(HN)$, and the amount of memory passed to the GPU is $O(VN)$, where V is the number of triangle mesh vertices. It is important to sum the contributions from all waves at a grid point if we want a detailed animation, but we can still reduce the number of interpolations and data accesses by lowering V (coarser triangle mesh), N (fewer pre-computed wavefronts), or H (resolution of the displayed height-field). Our dispersion approximation helps here by maintaining many wave details while reducing V and N. We also adaptively reduce N by deleting

Animation	H	V	W	M	$W \times M$	Memory	Precomputation	Runtime
Pond (see video)	1280×720	504	1	53	53	57kB	6 sec S	64 frames/sec
Puddle (see video and Figure 9)	1280×720	28404	16	9	144	28MB	5 min S	14 frames/sec
Ocean (Figure 1)	1280×4320	4	36	400	14400	5kB	30 sec S	0.91 frames/sec
Editing (Figure 10)	1280×1440	26259	1	15	15	19MB	1 min S	25 frames/sec
Interactive Island (see video)	640×1440	6004	18	12	216	57MB	30 min S / 5 min P	10 frames/sec
High Detail Island (Figure 2)	1280×2880	6004	18	200	3600	57MB	30 min S / 5 min P	0.21 frames/sec

Table I. : Performance details for our animations. H, V, W, and M represent the number of evaluated wave heights, triangle mesh vertices, tracked wavefronts, and dispersion-approximation waves per wavefront, respectively. $W \times M$ represents the total number of waves in the simulation, and "Memory" lists the memory required to store the multi-valued parameter functions. The "Precomputation" column denotes the time required to track waves in serial with 'S', and the time to track waves in a parallel batch across 6 processors with more aggressive adaptive time-stepping is denoted with a 'P'. Runtimes include high-quality adaptive rendering time; simplified rendering on a fixed grid speeds up runtimes significantly, as illustrated in §10.2.

wavefronts with low energy during precomputation, and we could potentially reduce the number of wave height evaluations M far away from the camera using the low-pass filtering procedure of [Hinsinger et al. 2002]. We were able to preview all of our examples at real-time rates (more than 60 frames per second), and then add additional details (by increasing H and M) for the final animation. We list details in Table I.

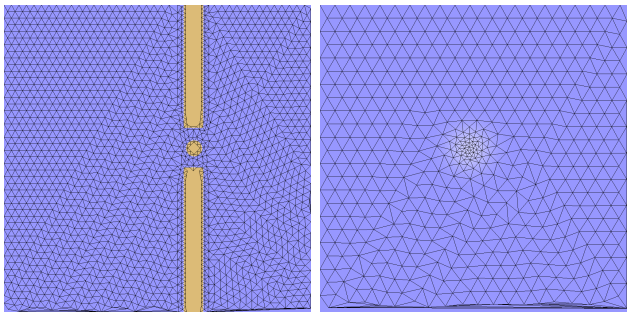
10.2 Comparison to an Eulerian method

It is difficult to directly compare our method with an Eulerian discretization, due to the completely different behavior of numerical parameters. Nevertheless, we found it informative to compare our method to the *iWave* simulation algorithm [Tessendorf 2004a], an Eulerian scheme known for its ability to recreate the four desired phenomena of refraction, dispersion, reflection, and diffraction. We ran simulations using both methods for similar water wave scenarios, and the experiments help to highlight the strengths and weaknesses of each approach. *iWave* has several numerical parameters such as artificial damping, time step size, and simulation resolution. We followed the parameter suggestions in [Tessendorf 2004a] where they were provided, and we tuned the remaining parameters to optimize simulation quality. This comparison uses a 2048^2 resolution, a timestep size of 0.001, and a damping parameter of 0.1, and it executes three simulation timesteps per frame of animation. For simulations using our method, we only tracked a few wavefronts on the coarse triangle meshes in Figure 11, and we used 100 dispersion waves. The small number of wavefronts decreases our precomputation cost at the expense of less detail, while an overly coarse triangle mesh has no practical performance benefit and is primarily used to highlight artifacts caused by our approach.

We attempted to set up two scenarios identically for both methods, but this is not straightforward and there are some differences in the initial conditions. Both simulations in Figure 12 feature a dispersive wavefront propagating from the left and then interacting with the environment. The wavefronts in our method were initialized with random wavelengths ranging from 1/2048 to 1/128 of the domain width. *iWave* is initialized by an oscillating step function whose Fourier transform should feature all wavelengths up to the minimum defined by the grid cell size (1/2048 of the domain width). GPU acceleration was used for both methods. Please see our additional supplemental video for motion comparisons between the methods.

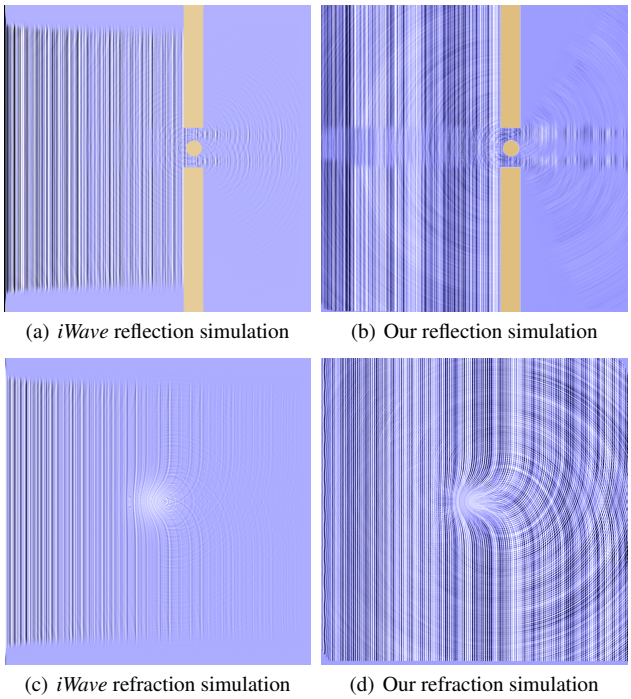
Reflections and diffractions. The top of Figure 12 simulates a series of waves reflecting and diffracting in an obstacle-dominated environment with constant depth. The *iWave* simulation clearly displays dispersion, reflects off the boundaries, and shows natural wavelength-dependent diffraction patterns. Although we tuned parameters to minimize numerical damping, the waves clearly lose energy as they travel long distances; decreasing the damping parameter any further causes the simulation to blow up. *iWave* requires customized non-reflecting boundary conditions by progressively damping near outer boundaries, but this was not a practical problem other than several cells of wasted resolution. [Tessendorf 2004b] states that the accuracy of the reflection method is unclear, and we only noticed such reflection inaccuracies (manifesting as a wavelength-biased numerical damping at the boundary) in a few rare situations. As an Eulerian method on a regular grid, *iWave* is well-suited for parallelism on the GPU, especially for the rectangular domains in this example. While *iWave* requires minimal precomputation in principle, steady-state scenarios like this one do require a significant amount of simulation time for waves to propagate all the way through the scene; the *iWave* simulation required 3 minutes and 15 seconds of pre-simulation to fill the domain with waves. For the simple 2D rendering in Figure 12 (1000^2 pixels), *iWave* ran at 162 frames per second. For the higher-quality adaptive 3D rendering of the same scene in our accompanying video (piecewise-bicubic interpolation of wave heights onto an adaptive 1280×2880 height-field), *iWave* ran at 11 frames per second.

For comparison, the simulation generated by our method used a single tracked wavefront and a triangle mesh with 2188 vertices. Our method exhibits more visible wave details than *iWave* and exhibits perfect energy conservation (until small waves are removed for efficiency reasons). It also reflects all wave frequen-



(a) Our reflection mesh; 3781 triangles, 2188 vertices (b) Our refraction mesh; 1022 triangles, 531 vertices

Fig. 11: Triangle meshes used by our method for the comparisons in §10.2.



(a) *iWave* reflection simulation (b) Our reflection simulation

(c) *iWave* refraction simulation (d) Our refraction simulation

Fig. 12: We compare our method to *iWave* for scenarios which isolate reflection and diffraction (top) and refraction (bottom). To highlight differences, we show a 2D top-down view here. 3D fly-through animations can be found in our additional supplemental video.

cies equally by construction. Our method does not diffract waves as accurately as *iWave*, because our diffraction algorithm is currently independent of wavelength. This example also shows the visual artifacts that can result from a coarse triangle mesh. The steep change in amplitude caused by this double-slit scenario is interpolated across very coarse triangles, which smears out the amplitude gradient in a mesh-dependent manner. A finer mesh, additional wavefronts from slightly different angles, a more natural environment, or some damping would alleviate this problem. While our method parallelizes well on the GPU, load-balancing is more variable than a simple Eulerian method; complex regions with more overlapping waves require more interpolations, while simpler areas require fewer. Our simulation required 25 seconds of precomputation, the 2D rendering ran at 107 frames per second, and the 3D rendering ran at 8 frames per second.

Refraction. The bottom of Figure 12 simulates a series of waves refracting over an underwater bump. *iWave* clearly displays dispersion and refraction behaviors as expected, and the balance between damping and stability is still a problem for waves traveling over long distances. *iWave* shows how a steep change in water depth can spawn both reflected and refracted waves. The *iWave* simulation required 3 minutes and 15 seconds of pre-simulation to fill the domain with waves, the 2D rendering ran at 160 frames per second, and the high-quality 3D rendering ran at 11 frames per second.

Our method used three tracked wavefronts and a triangle mesh with 531 vertices for this example. Our resulting simulations exhibit longer lasting waves and sharper details, but it does not both reflect and refract waves when it crosses the shallow area. This is because we must explicitly spawn reflected waves when they hit obstacles; refraction is the default behavior. Mesh-dependent artifacts are not visible despite the coarse triangle mesh, because there are no unnaturally steep changes in amplitude. Our simulation required 14 seconds of precomputation, the 2D rendering ran at 124 frames per second, and the 3D rendering ran at 8 frames per second.

Comparison performance notes. Both methods had similar runtimes for these chosen examples. Again, it is hard to directly compare the methods due to incompatible numerical parameters, but we found that the number of height-field evaluations at runtime was the bottleneck for both methods. In each of our experiments, both methods gained an order of magnitude speedup from 3D to 2D by using a simpler shader and rendering fewer height-field samples (reducing H). Both our method and *iWave* can sacrifice visual quality for speed by using fewer samples in the displayed heightfield, using simple linear interpolation of wave heights, or directly rendering wave heights on the simulation grid/mesh. Another thing to note is that larger simulation resolutions will eventually slow down an Eulerian method by requiring even more simulation time steps per frame of animation, while our method can be evaluated at any point in time, independent of mesh resolution.

11. DISCUSSION

While this work represents several significant advances in ocean animation, there is still work to be done. The method is currently limited to static obstacles and pre-computed wave paths; it is not yet able to accurately handle moving boundaries or interactive changes in wave direction. The precomputation phase of our algorithm can also be viewed as a limitation when compared to other methods, though we are able to reduce this drawback using parallelization and adaptive ODE integration.

Our approach successfully reconstructs the high-frequency η function by sampling the ϕ and a functions instead of η directly. To avoid under-sampling these functions, we maintain a finer triangle mesh resolution near curved obstacles and sharp changes in water depth, where we expect sharper geometric curvature of the wavefronts and thus larger derivatives of ϕ. Higher-order interpolation methods almost eliminate these errors, even with coarse meshes. In contrast to Eulerian methods, re-meshing our triangle mesh is *not* analogous to changing simulation resolution; it is more like resampling a function at different locations. Higher mesh resolution resolves finer variations in the phase function, but it will not change the overall behavior of wave paths, shapes, wavelengths, or the timing of simulation events.

The underlying linearized wave model prevents any overturning waves or rotational flows, just like previous procedural wave models. While the eikonal high-frequency approximation only guarantees accuracy for wavelengths smaller than the scale of variations in the environment, we found it difficult to detect visual artifacts (like wavelength-dependent diffraction) even for large wavelengths.

On the positive side, our algorithm is capable of generating multitudes of ocean waves that interact with their environment and exhibit high-resolution details at efficient frame rates. Our method is easily parallelizable and maps well to GPU hardware. While the approach is not as versatile as fully 3D Navier-Stokes simulations, it is virtually independent of the Nyquist and CFL restrictions that plague other water simulation techniques. In fact, our method is unconditionally stable at runtime, and it does not exhibit any numerical damping. Our technique also handles non-reflecting boundary conditions by default, and it has the unique feature that visible wave detail is independent of simulation complexity. Unlike Eulerian grid methods, the analytical wave nature of our method allows us to use Gerstner or Biesel wave profiles to model detailed tan-

gential wave motions. We see our method as a generalized version of traditional spectrum-based ocean simulation techniques, which are currently the state of the art in the simulation of detailed ocean textures and large bodies of water.

ACKNOWLEDGMENTS

We would like to thank Nafees Bin Zafar for an informative discussion about the motivating problem and David Hahn for proofreading several drafts of this work. Most importantly, we would like to thank the reviewers for offering their insight and helping improve our paper.

REFERENCES

AIRY, G. B. 1841. *Tides and waves*. London.

ANDO, R., THUEREY, N., AND WOJTAN, C. 2013. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. Graph. 32,* 4 (July), 103:1–103:10.

BULANT, P. AND KLIMEŠ, L. 1999. Interpolation of ray theory traveltimes within ray cells. *Geophys. J. Int. 139,* 2, 273–282.

CHENTANEZ, N. AND MÜLLER, M. 2010. Real-time simulation of large bodies of water with small scale details. In *Proc. ACM SIGGRAPH/Eurographics Symp. on Comp. Anim.* 197–206.

DARLES, E., CRESPIN, B., GHAZANFARPOUR, D., AND GONZATO, J.-C. 2011. A survey of ocean simulation and rendering techniques in computer graphics. In *Comput. Graph. Forum*. Vol. 30. 43–60.

DEAN, R. G. AND DALRYMPLE, R. A. 1991. *Water wave mechanics for engineers and scientists.* World Scientific.

FOSTER, N. AND FEDKIW, R. 2001. Practical animation of liquids. In *Proc. SIGGRAPH 01*. Annual Conference Series. 23–30.

FOURNIER, A. AND REEVES, W. T. 1986. A simple model of ocean waves. In *Computer Graphics*. Vol. 20. ACM, 75–84.

GAMITO, M. N. AND MUSGRAVE, F. K. 2002. An accurate model of wave refraction over shallow water. *Computers & Graphics 26,* 2, 291–307.

GONZATO, J.-C. AND LE SAËC, B. 1997. A phenomenological model of coastal scenes based on physical considerations. In *Computer Animation and Simulation '97*. 137–148.

GONZATO, J.-C. AND LE SAËC, B. 2000. On modelling and rendering ocean scenes. *J. Vis. and Comput. Anim. 11,* 1, 27–37.

HINSINGER, D., NEYRET, F., AND CANI, M.-P. 2002. Interactive animation of ocean waves. In *Proc. ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* 161–166.

KASS, M. AND MILLER, G. 1990. Rapid, stable fluid dynamics for computer graphics. In *Computer Graphics*. Vol. 24. 49–57.

KEELER, T. AND BRIDSON, R. 2014. Ocean waves animation using boundary integral equations and explicit mesh tracking. In *Proceedings of the 13th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '14. Eurographics.

LAMB, H. 1895. *Hydrodynamics*. Cambridge University Press.

LEVY, B. R. AND KELLER, J. B. 1959. Diffraction by a smooth object. *Comm. Pure and Appl. Math. 12,* 1, 159–209.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. 23,* 3, 457–462.

MASTIN, G. A., WATTERBERG, P. A., AND MAREDA, J. F. 1987. Fourier synthesis of ocean scenes. *Computer Graphics and Applications, IEEE 7,* 3, 16–23.

MCCANN, J. AND POLLARD, N. S. 2008. Real-time gradient-domain painting. *ACM Trans. Graph. 27,* 3 (Aug.), 93:1–93:7.

MOORE, M. AND WILHELMS, J. 1988. Collision detection and response for computer animation. In *Computer Graphics*. Vol. 22. ACM, 289–298.

NIELSEN, M. AND BRIDSON, R. 2011. Guide shapes for high resolution naturalistic liquid simulation. In *ACM Trans. Graph.* Vol. 30. ACM, 83.

NIELSEN, M. B., SÖDERSTRÖM, A., AND BRIDSON, R. 2013. Synthesizing waves from animated height fields. *ACM Trans. Graph. 32,* 1, 2.

NIELSON, G. M. 1979. The side-vertex method for interpolation in triangles. *Journal of Approximation Theory 25,* 4, 318 – 336.

O'BRIEN, J. F. AND HODGINS, J. K. 1995. Dynamic simulation of splashing fluids. In *Proc. Comp. Anim. '95*. IEEE, 198–205.

PADRINO, J. C. AND JOSEPH, D. D. 2007. Correction of Lamb's dissipation calculation for the effects of viscosity on capillary-gravity waves. *Physics of Fluids 19*, 082105.

PEACHEY, D. R. 1986. Modeling waves and surf. In *Computer Graphics*. Vol. 20. ACM, 65–74.

RUNBORG, O. 2007. Mathematical models and numerical methods for high frequency waves. *Comm. Comp. Phys 2,* 5, 827–880.

SETHIAN, J. A. 1999. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*. Vol. 3. Cambridge university press.

SHEWCHUK, J. R. 1996. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Applied computational geometry towards geometric engineering*. Springer, 203–222.

SIDEFX. 2013. Houdini 13.0 wave layer tank. http://www.sidefx.com/docs/houdini13.0/shelf/wavelayertank.

SÖDERSTRÖM, A., KARLSSON, M., AND MUSETH, K. 2010. A pml-based nonreflective boundary for free surface fluid animation. *ACM Trans. Graph. 29,* 5 (Nov.), 136:1–136:17.

TESSENDORF, J. 2004a. Interactive water surfaces. *Game Programming Gems 4*, 265–274.

TESSENDORF, J. 2004b. Simulating ocean water. *ACM SIGGRAPH Courses*.

THUEREY, N., MULLER-FISCHER, M., SCHIRM, S., AND GROSS, M. 2007. Real-time breaking waves for shallow water simulations. In *Proc. Pacific Graphics*. IEEE, 39–46.

THUEREY, N., SADLO, F., SCHIRM, S., MÜLLER-FISCHER, M., AND GROSS, M. 2007. Real-time simulations of bubbles and foam within a shallow water framework. In *Proc. ACM SIGGRAPH/Eurographics Symp. on Comp. Anim.* 191–198.

TS'O, P. Y. AND BARSKY, B. A. 1987. Modeling and rendering waves: wave-tracing using beta-splines and reflective and refractive texture mapping. *Computer Graphics 6,* 3, 191–214.

URSIN, B. 1982. Quadratic wavefront and traveltime approximations in inhomogeneous layered media with curved interfaces. *Geophysics 47,* 7, 1012–1021.

WHITTED, T. 1980. An improved illumination model for shaded display. *Commun. ACM 23,* 6 (June), 343–349.

YUKSEL, C., HOUSE, D. H., AND KEYSER, J. 2007. Wave particles. *ACM Trans. Graph. 26,* 3, 99.

APPENDIX

A. DIFFRACTION EXPERIMENT

As mentioned in §5.1, in the special case of constant wavenumber k and boundary curvature κ, Equation 11 states that α is constant. The integral in Equation 11 then reduces to the quantity $-s\alpha$. Thus, diffractions around perfect circles should decay exponentially with the geodesic distance along the boundary, and we can measure the constant decay rate by computing the change in the logarithm of amplitude $\alpha = -d(\log(a))/ds$. We know that $\alpha \propto \kappa^{2/3}$, so $\alpha^{-3/2}$ is proportional to the radius of curvature of the circle, $1/\kappa$.

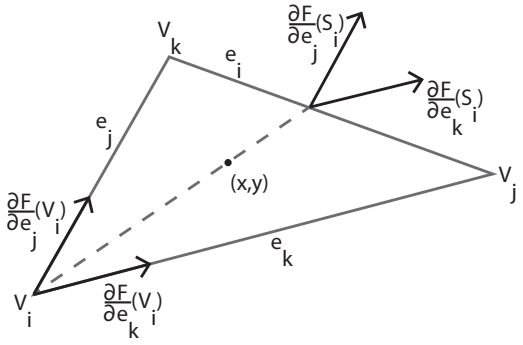


Fig. 13: The geometric setup for the side-vertex interpolation scheme.

We ran several simulations in which waves diffract around circles of varying radii, and we plotted the $\alpha^{-3/2}$ values as a function of boundary curvature in Figure 5. The geometric theory of diffraction predicts that these two quantities must be linearly related, and indeed linear regression analysis yields an excellent fit of $R^2 = 0.9721$.

B. DERIVATIVES AND INTERPOLATION

As described in §6.1, our interpolation scheme requires first derivatives at each triangle mesh vertex, which are calculated during the wave propagation in our precomputation phase. The first derivative of ϕ can be computed from the eikonal equation:

$$\nabla\phi = \vec{\mathbf{c}}/c^2 \tag{15}$$

where $\vec{\mathbf{c}}$ is the phase velocity vector. The first derivative of amplitude can be computed using the gradient of bilinear interpolation of the wavefront segments. Second derivatives of ϕ can be evaluated similarly if necessary for higher-order interpolation.

Ursin [1982] shows that ϕ^2 can be interpolated more accurately than ϕ. As an intuitive example, a circular wavefront with constant phase speed will produce the cone $\phi{=}\sqrt{x^2+y^2}/c$, which is not polynomial and exhibits a singularity in the first derivative. However, squaring it gives $\phi^2{=}(x^2+y^2)/c^2$, which can be reconstructed exactly with a quadratic interpolation scheme. To adapt this idea to our model, we compute ϕ as usual during the wavefront propagation. Then, at runtime, we set $\Phi{=}\phi^2$, interpolate Φ instead of ϕ, and then finally use $\phi{=}\sqrt{\Phi}$ when needed.

For high-order interpolation of Φ, its derivatives can be computed in terms of the already-known derivatives of ϕ:

$$\nabla\Phi = 2\phi\nabla\phi \tag{16}$$

C. SIDE-VERTEX INTERPOLATION SCHEME

This section lists the equations required to implement the side-vertex interpolation scheme [Nielson 1979], as mentioned in §7. Please refer to Figure 13 for an illustration of the geometric setup.

Input: Triangle with vertices V_i and per-vertex values $F(V_i)$, partial derivatives $F_x(V_i), F_y(V_i)$, as well as a position (x, y) inside the triangle with barycentric coordinates (b_1, b_2, b_3).

Output: C^1 smoothly interpolated function value $D[F]$ at the given position (x, y).

$$D[F] = \frac{b_2^2 b_3^2 D_1[F] + b_1^2 b_3^2 D_2[F] + b_1^2 b_2^2 D_3[F]}{b_2^2 b_3^2 + b_1^2 b_3^2 + b_1^2 b_2^2} \tag{17}$$

with

$$\begin{aligned} D_i[F] &= B_i[F] + P_i[F] \\ i = 1,2,3; \qquad & i \neq j \neq k \neq i \end{aligned} \tag{18}$$

where

$$\begin{aligned} B_i[F] =& h(1-b_i)F(S_i) \\ & - b_i(1-b_i)\left[b_j \frac{\partial F}{\partial e_k}(S_i) + b_k \frac{\partial F}{\partial e_j}(S_i)\right] \end{aligned} \tag{19}$$

$$P_i[F] = h(b_i)F(V_i) + b_i^2\left[b_j \frac{\partial F}{\partial e_k}(V_i) + b_k \frac{\partial F}{\partial e_j}(V_i)\right] \tag{20}$$

$$h(t) = t^2(3-2t) \tag{21}$$

$$\frac{\partial F}{\partial e_j}(V_i) = (x_k - x_i)F_x(V_i) + (y_k - y_i)F_y(V_i) \tag{22}$$

$$\frac{\partial F}{\partial e_j}(S_i) = (x_k - x_i)F_x(S_i) + (y_k - y_i)F_y(S_i) \tag{23}$$

Along each triangle edge, the function value $F(S_i)$ is computed using Hermite interpolation from the vertices. The tangential derivative is computed as the directional derivative of this Hermite function along the edge. The normal derivative along the edge is computed by interpolating the normal derivatives from the vertices using a Hermite spline with zero second derivatives. The tangential and normal derivatives are then converted to the partial derivatives in cartesian coordinates F_x and F_y. Please see the original paper by Nielson [1979] for more details.

D. DISPERSION APPROXIMATION ERROR

As discussed in §8.1, we aim to compute ϕ_{ij} and a_{ij} of a new wave ij that is nearly identical to wave i except for its wavenumber $k_{ij}{\neq}k_i$. Equation 4 can solved for ϕ by integrating along the wavefront travel path $\mathcal{P}$:

$$\phi_i = \int_{\mathcal{P}} \frac{ds}{c(k_i)} \tag{24}$$

We wish to express ϕ_{ij} in terms of ϕ_i:

$$\begin{aligned} \phi_{ij} &= \int_{\mathcal{P}} \frac{ds}{c(k_{ij})} \\ &= \int_{\mathcal{P}} \frac{c(k_i)}{c(k_{ij})} \frac{ds}{c(k_i)} \\ &= \frac{c(k_i)}{c(k_{ij})} \int_{\mathcal{P}} \frac{ds}{c(k_i)}, \qquad \left(\text{Assuming} \frac{c(k_i)}{c(k_{ij})} \text{is constant}\right) \\ &= \frac{c(k_i)}{c(k_{ij})}\phi_i \end{aligned} \tag{25}$$

Thus we can avoid numerical integration of ϕ_{ij} altogether if we approximate $c(k_i)/c(k_{ij})$ as constant. For simplicity, we choose to set $C_{ij}{=}c(k_i)/c(k_{ij})$ based on the initial phase speeds.

The most obvious visual artifact of this approximation is that wave ij must follow the same path as wave i, so the waves cannot separate due to wavelength-dependent refraction. However, the approximation is exact in the common regimes of deep water, capillary waves, and constant depth (which do not experience refraction), as well as for gravity waves in shallow water (where refraction is independent of wavelength). However, while C_{ij} is indeed constant within these regimes, the value of C_{ij} will vary from one

regime to another. Thus, the approximation is less accurate when waves transition between regimes. The error clearly converges to zero as $k_{ij} \to k_i$, so smaller deviations from k_i will have fewer artifacts.

The error in assuming $a_{ij} = A_{ij} a_i$ is coupled to the ϕ_{ij} error, but the analysis is more difficult (because a depends on both D and c_g). We found that amplitude variations are not as obvious as changes in wave direction, and we did not perceive any visual artifacts from this amplitude approximation.

Received September 2014; accepted January 2015