EUROGRAPHICS 2002 / I. Navazo Alvaro and Ph. Slusallek
(Guest Editors)

A Convolution-Based Algorithm for Animated Water Waves

J. Loviscach

Fachbereich Elektrotechnik und Informatik, Hochschule Bremen, Bremen, Germany

Abstract

A non-linear partial differential equation solver is too sophisticated for computer graphics applications if they are only used to render effects like circular waves and ship wakes. We present an approach which simulates waves through a convolution algorithm. It handles both gravity waves and capillary waves; the latter are often neglected even though they dominate small-scale behavior. The algorithm can be integrated into a complete solution architecture: First, standard commercial 3-D software is used to prepare an animated scene with objects traveling on a water surface. Based on the movements of these objects, waves are calculated and added as bump and displacement maps to the 3-D model.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Water waves are an everyday phenomenon and thus their simulation makes for a natural research topic in computer graphics. Despite all past efforts, both the analytical and numerical treatment of a liquid's motion still pose considerable challenges. Fluid motion is described by the continuity equation plus the Navier-Stokes equations—a system of three non-linear partial differential equations—in conjunction with surface conditions, which typically are non-linear as well. However, a full-blown numerical solution of these equations is rarely needed to generate realistically-looking animated films.

Since the motion in the volume of the liquid remains invisible in most scenes, it is sufficient to model the surface of the liquid. Additionally, water is virtually incompressible, its viscosity can often be neglected, and non-linear effects stay small if the waves' height is small when compared to their length. When using an approximate, linear system of partial differential equations, one arrives at the conclusion[2] that small-amplitude surface water waves on an flat infinite ocean with infinite depth are composed of sinusodial waves

$$(\mathbf{x}, t) \mapsto \cos(\mathbf{k} \cdot \mathbf{x} - \omega(\mathbf{k})t + \phi),$$

where $\mathbf{x} \in \mathbb{R}^2$ is the position on the plane of the ocean, $\mathbf{k} \in \mathbb{R}^2$ the wave vector along the surface, $\omega(\mathbf{k})$ the corresponding circular frequency, and $\phi \in \mathbb{R}$ the phase. The cir-

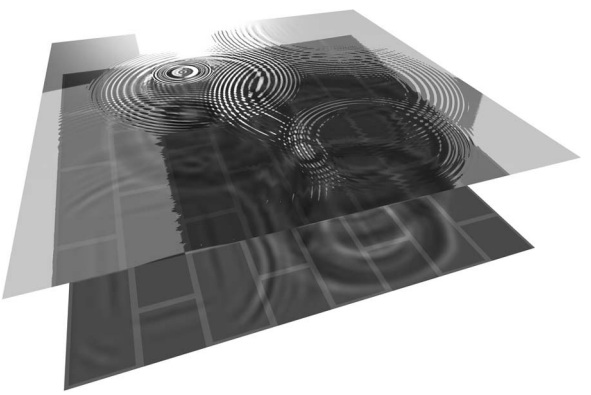

Figure 1: *A raindrop simulation by the proposed algorithm shows the characteristic multiple rings formed by the interplay of capillary and gravity waves. The results of the simulation have been used for a bump map and a caustics texture in a rendering created with Cinema 4D XL. Compare with figure 3 for the height field.*

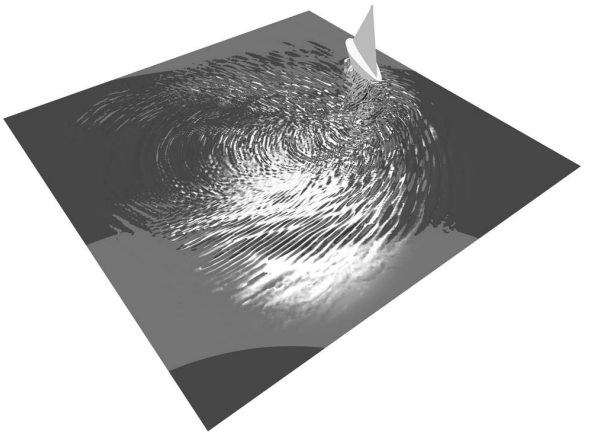

Figure 2: *By calculating appropriate convolution kernels for each swimming object at each time step, the algorithm is able to handle arbitrary ship paths. Here, its results are used as bump and displacement maps with Cinema 4D XL. The height field can been seen in figure 8.*

cular frequency only depends on the wave number $k = |\mathbf{k}|$ and is given by

$$\omega(k) = \sqrt{gk + \frac{T}{\rho}k^3}, \tag{1}$$

with gravity acceleration g, surface tension T, and mass density ρ. Under normal circumstances, their values are $g = 9.8\,\mathrm{kg\,m/s^2}$, $T = 0.072\,\mathrm{kg/s^2}$, and $\rho = 1.0 \cdot 10^3\,\mathrm{kg/m^3}$.

The dispersion relation $\omega(k)$ causes a number of important visual effects:

- The phase velocity $\omega(k)/k$ reaches its minimum of $23\,\mathrm{cm/s}$ at a wavelength of $17\,\mathrm{mm}$. Shorter waves (capillary waves dominated by surface tension) as well as longer waves (gravity waves) travel faster. Likewise, the group velocity $d\omega(k)/dk$ possesses a minimum of $18\,\mathrm{cm/s}$. A stone thrown into deep water produces complex ring waves where the innermost ring expands at this speed. In contrast to this behavior of water surface waves, waves of light or sound travel at a constant speed, regardless of their frequency. If water behaved this way, a stone thrown into water would yield a single sharp circular wave.
- A ship traveling at constant speed along a straight line on deep water leaves behind a Kelvin wake[2]—a wedge with semiangle 19°. This angle is independent of the ship's velocity, once more in stark contrast to sound waves: The Mach cone of a supersonic plane gets more pointed as the speed of the aircraft increases.

Therefore, an approach using the much simpler wave equation of sound and light often results in unrealistic behavior. Nonetheless, this equation is favored e. g. in computer games because of its simple solvability[8]. In order to realistically simulate ring waves and Kelvin ship waves for 3-D animated films without too much mathematical and computational overhead, we sought a simple solution to the linear approximation of water surface waves outlined above.

Due to the complicated dispersion relation, there is no simple finite-difference formulation for this approximation. However, one can describe the propagation of waves with help of convolution. Let the elevation of the wave field be $\eta_t(\mathbf{x})$ with $\eta = 0$ representing the surface at rest. Store the momentary state of the ocean at time t as a complex-valued function $\mathbf{x} \mapsto \eta_t(\mathbf{x}) \in \mathbb{C}$ and use the real value of this function for display. Given η_t, one can calculate the elevation at any time $t + \Delta t$ by convolution with a certain kernel function $\Psi_{\Delta t}(\mathbf{x})$:

$$\eta_{t+\Delta t}(\mathbf{x}) = \iint_{\mathbb{R}^2} d^2\mathbf{y}\, \Psi_{\Delta t}(\mathbf{y})\, \eta_t(\mathbf{x}-\mathbf{y}). \tag{2}$$

Formally, the kernel Ψ is given by the mathematically ill-defined expression

$$\Psi_{\Delta t}(\mathbf{x}) = \iint_{\mathbb{R}^2} \frac{d^2\mathbf{k}}{4\pi^2} \exp\Big(i(\mathbf{k}\cdot\mathbf{x} - \omega(\mathbf{k})\Delta t)\Big). \tag{3}$$

Like in Huygen's principle, this is the circular wave caused by a disturbance at time 0 at the origin, observed after a time span of Δt. In order to use this kernel with a lattice-based numerical approximation, it is necessary to introduce cutoffs for long and short waves. These cutoffs have the additional benefit of eliminating the mathematical problems with this expression.

The convolution scheme allows to calculate each frame without intermediate time steps. Sudden disturbances, such as a stone dropped into a pool, can be incorporated by altering η_t at the time t of this incident, for example by adding a function to η_t which is zero outside the cross section of the stone. In order to model continuous disturbances such as traveling ships, however, one has to integrate over the time-span Δt.

We have developed a prototype system for water wave animation with this algorithm. The solution is based on Maxon Cinema 4D XL, but could use virtually any other commercial 3-D software package. This program is used to model and animate a flat ocean and objects traveling on the water, which still is flat. These pictures are rendered black and white in a view form above and then fed into our special add-on software. It computes the corresponding wave elevation fields, optionally separates the results into a blurred, low-pass version and a corresponding high-pass version. In addition, the software calculates the caustics pattern on the ground. These parallel sequences of pictures are fed into the 3-D software as displacement maps, bump maps, and texture maps for the final rendering—now including waves and seen from an arbitrary point of view. The division into displacement maps and bump maps allows for a relatively coarse tesselation resulting in short rendering times.

The next section outlines previous work done for simulating water waves in 3-D computer graphics. Section 3 describes our convolution-based algorithm for time evolution. Section 4 explains how objects which move on the water surface are incorporated into this approach. A discussion of the results follows in section 5. The final section 6 closes with a summary and an outlook on possible extensions.

2. Previous Work

The first two major works about water wave animation rooted in physics have been published in 1986: Fournier and Reeves[5] as well as Peachey[14] use wave trains with the shape of a trochoid respectively an approximation to a trochoid shape. Waves running onto a shore are subjected to refraction and therefore compressed. Gonzato[7] carries the idea of "wave tracing" further to calculate refraction due to varying water depth and the diffraction at obstacles. Here, capillary waves are introduced, even though they are simply modeled by fractal noise. In their recent work, Gamito and Musgrave[6] offer a more detailed theory of wave tracing.

Thon and Ghazanfarpour[18] use wave spectra measured e. g. from photographs to synthesize ocean waves. Using quite a different approach, Schneider and Westermann[15] simulate waves using fractal noise and then render these in real time with light reflection, refraction, and caustics with modern consumer graphics hardware.

Goss[9] presents an particle-system algorithm tailored to display ship wakes in real-time. Khan[10] superimposes circular waves from points along a ship's path to simulate its wake.

To allow a more complete interaction of the water with disturbances as well as changing boundaries, there have been several approaches based on partial differential equations. Kaas and Miller[11] use a linearized model of a shallow water limit, which is equivalent to the wave equation of light and sound. This is solved by a finite-difference-method. Recently, Layton and van de Panne[12] have improved these ideas by incorporating non-linear advection.

O'Brien and Hodgins[13] model the water volume as an array of square columns which exchange fluid. If the content of a column moves upward faster than a threshold velocity, their algorithm generates spray. To achieve real-time response, Chen and Lobo[1] replace the full three-dimensional Navier-Stokes equations by the two-dimensionial version. The local pressure calculated in this manner is used to steer a three-dimensional height field which represents the water surface.

Foster and Metaxas[3] simulate solutions of the full Navier-Stokes equations by using a voxel representation for the volume and marker particles to trace the surface. Thanks to this representation, their algorithm can produce e. g. breaking waves and overturns. Surface tension is simulated by altering pressure gradients. To stabilize the method for large time steps, a damping term has to be introduced. Taking a different road, Stam[16] proposes an intrinsically stable algorithm for the full Navier-Stokes equations. It can be simplified using Fourier transform[17]. However, the presented algorithm does not allow free boundaries such as water surfaces. Foster and Fedkiw[4] combine Stam's solver with marker particles[3] to trace the fluid surface.

Weimer and Warren[19] note that certain multi-grid partial differential solvers can be formulated with help of vector subdivision operations. The method they describe assumes a slow flow, i. e. viscosity dominates inertial behavior, and is therefore not well-suited for water simulation.

3. Time Evolution by Convolution

To treat the problem numerically, we discretize the elevation field η_t to a finite lattice of size $L \times L$ with $N \times N$ points. It consists of the $(p\Delta x, q\Delta x)$ with $p,q = 0,1,\ldots,N-1$ and $\Delta x = L/(N-1)$. In our experiments, $N = 256$ or, in extreme cases, 512 proved to be sufficiently large. The real part of the computed η_t will be delivered as a bump map and/or displacement map to the 3-D rendering software. Therefore, such maps will then consist of e. g. 256×256 pixels.

To calculate the elevation field in an animated film from one picture frame to the next, we use the time evolution given by equation 2 with Δt being the time between frames. The double integral is approximated using the double sum

$$\tilde{\eta}_{t+\Delta t}(p\Delta x, q\Delta x) = \qquad (4)$$
$$(\Delta x)^2 \sum_{r=-M}^{M-1} \sum_{s=-M}^{M-1} \Phi_{\Delta t}(r\Delta x, s\Delta x)\, \eta_t((p-r)\Delta x, (q-s)\Delta x).$$

Here, η_t is treated as zero outside of $[0,L] \times [0,L]$. M measures the half size of a sliding window which is used for the convolution. We use $M = 40$, so that the window contains 80×80 lattice points. Instead of the ill-defined $\Psi_{\Delta t}$ of equation 3 a regularized version $\Phi_{\Delta t}$ is used. Especially, $\Phi_{\Delta t}$ has to smoothly approach zero on the boundary of the sliding window (see below).

The $\tilde{\eta}_{t+\Delta t}(p\Delta x, q\Delta x)$ resulting from equation (4) cannot be used directly, because the resulting waves would reflect strongly at the borders. This reflection can be suppressed reliably by cutting off the elevation field near the borders in linear fashion:

$$\eta_{t+\Delta t}(p\Delta x, q\Delta x) = h(p)\, h(q)\, \tilde{\eta}_{t+\Delta t}(p\Delta x, q\Delta x) \qquad (5)$$

with

$$h(p) := \begin{cases} p/M & \text{if } p < M \\ 1 & \text{if } M \le p < N-M \\ (N-1-p)/M & \text{if } p \ge N-M. \end{cases}$$

The resulting elevation field $\eta_{t+\Delta t}$ will be used to calculate the next frame's field $\eta_{t+2\Delta t}$, which in turn will serve as base for the succeeding one, etc.

The remaining task is to construct a regularized kernel

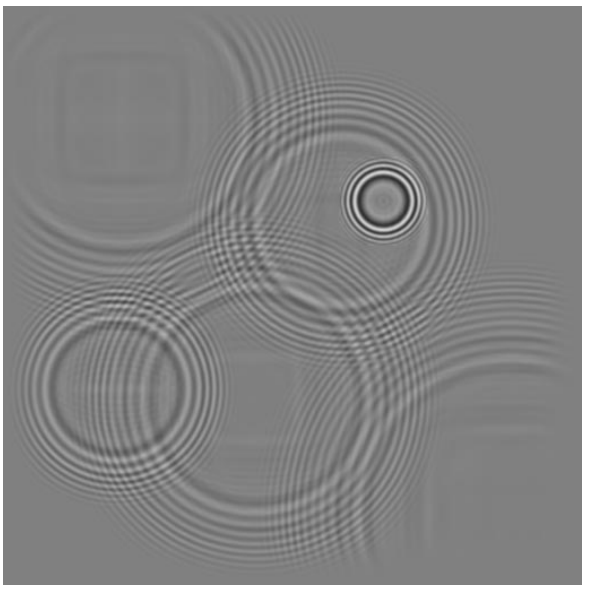

Figure 3: *Raindrop simulation by disturbing η_t at pseudo-random places and pseudo-random times. Parameters: $L = 1\,m$, $\Delta t = 0.2\,s$, $N = 512$, $2M = 80$. See figure 1 for a 3-D rendering. To save calculation time, a quite low M has been selected. Consequently, $\Phi_{\Delta t}$ is slightly cut off at the boundary of the sliding window (see figure 4). This causes rectangular artifacts inside the rings. In the 3-D rendering, however, these remain nearly invisible.*

$\Phi_{\Delta t}$. Since the time step Δt is constant throughout the animation sequence, $\Phi_{\Delta t}$ can be computed in advance, stored in an $2M \times 2M$ array of complex numbers and used for every picture frame.

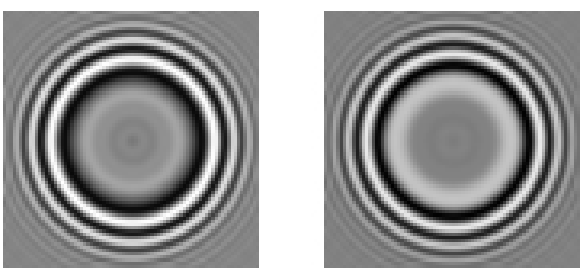

Figure 4: *The convolution kernel $\Phi_{\Delta t}$ used in calculating figure 3. The left picture shows the real part, the right picture the imaginary part. For better comprehensibility, they are enlarged in comparison to figure 3.*

A necessary property of $\Phi_{\Delta t}(p\Delta x, q\Delta x)$ is that it tends to zero at the boundary of its domain, i. e. as p or $q \to -M$ or $M-1$. This cannot simply be implemented by multiplying with a cutoff function in $\mathbf{x}$-space. Especially, one would sacrifice the fundamental propagation identity of Ψ:

$$\Psi_{s+t}(\mathbf{x}) = \iint_{\mathbb{R}^2} d^2\mathbf{y}\, \Psi_s(\mathbf{y})\, \Psi_t(\mathbf{x}-\mathbf{y}). \tag{6}$$

In order to preserve this property, we have chosen to modify the dispersion relation given in equation (1) by replacing

$$\omega(k) \to \omega(k) - ic(k), \tag{7}$$

where $c(k) \geq 0$ is a damping function depending on the wave number. The well-defined analogue of equation 3 then is a discrete Fourier transform of a function with fast decay.

To define $c(k)$, one has to identify the waves which are to be suppressed. Using the stationary phase method[2] to approximate equation (3), one finds that the major contributions to $\Psi_{\Delta t}(\mathbf{x})$ arise from wave vectors $\mathbf{k}$ at which the exponent has zero gradient, i. e. $\mathbf{x}/\Delta t$ equals the group velocity vector

$$\mathbf{v}_G(\mathbf{k}) := \frac{\partial\omega(\mathbf{k})}{\partial\mathbf{k}} = \frac{g + 3T|\mathbf{k}|^2/\rho}{2\omega(\mathbf{k})}\mathbf{k}^0. \tag{8}$$

We want to cutoff contributions to $\mathbf{x}$ which lie near the border of the $2M \times 2M$ window, or, to be specific, where $|\mathbf{x}| > \frac{3}{4}M\Delta x$. Hence, the $\mathbf{k}$ which follow the following criterion have to be supressed:

$$|\mathbf{v}_G(\mathbf{k})| > \frac{3}{4}\frac{M\Delta x}{\Delta t}.$$

To this end, one can choose $c(k)$ in the modified dispersion relation (7) as follows:

$$c(|\mathbf{k}|) := \frac{5}{M\Delta x}\left(|\mathbf{v}_G(\mathbf{k})| - \frac{3}{4}\frac{M\Delta x}{\Delta t}\right)_+ \tag{9}$$

where the pre-factor is somewhat arbitrary (this choice proved to be useful in our experiments) and where

$$(a)_+ := \begin{cases} a & \text{if } a > 0 \\ 0 & \text{else.} \end{cases}$$

To implement the cutoff, we found two other promising options which appear to merit further investigation: First, equation (9) could be reformulated using the smoother expression

$$\frac{100\Delta t}{(M\Delta x)^2}\left(|\mathbf{v}_G(\mathbf{k})| - \frac{1}{2}\frac{M\Delta x}{\Delta t}\right)_+^2.$$

Secondly, the group velocity could be kept below $v_{\max} := \frac{3}{4}\frac{M\Delta x}{\Delta t}$ by approximating the dispersion relation $\omega(k)$ of equation (1) with

$$\sqrt{gk + \frac{T}{\rho}k^2\, a \arctan(k/a) + b^2},$$

where

$$a := \frac{2\rho v_{\max}^2}{\pi T} \quad \text{and} \quad b := \frac{g}{2v_{\max}}.$$

4. Wave Generation by Swimming Objects

Ships and other objects in the water surface generate waves through their motion and/or form changes. Our solution uses the motion between two picture frames to calculate how the elevation field $\eta_{t+\Delta t}$ is to be altered for the following frame. For each swimming object, an additive contribution is computed.

For each object, we render the animation as a picture sequence where the scene is shown from straight above, with the ocean in black and the cross-section formed by the object with the water surface in white. Such a rendering can easily be obtained through Boolean operations available in standard 3-D animation software.

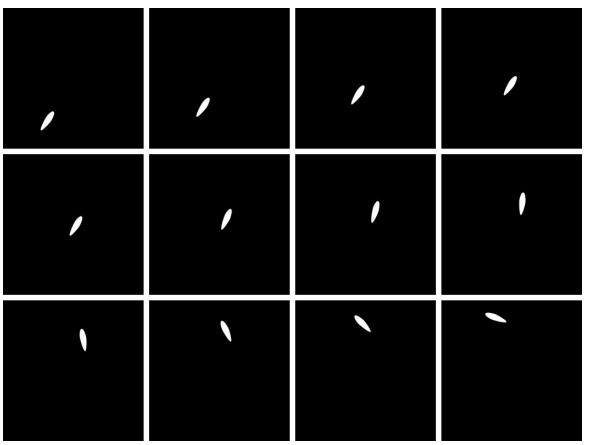

Figure 5: *A picture sequence of the Boolean intersection between ship and ocean rendered from above in black and white is used to determine the ship's momentary velocity vector and the function f_t. For the simulation result see figure 8, for a 3-D rendering see figure 2.*

The function $f_t(\mathbf{x})$ describes the greyscale picture rendered this way for time t, see figure 5. From f_t and $f_{t+\Delta t}$, we interpolate a corresponding moving and changing form for all intermediate times $s \in [t, t+\Delta t]$:

$$f_s(\mathbf{x}) := \frac{t+\Delta t-s}{\Delta t} f_t\left(\mathbf{x}-(s-t)\mathbf{v}\right) + \frac{s-t}{\Delta t} f_{t+\Delta t}\left(\mathbf{x}+(t+\Delta t-s)\mathbf{v}\right),$$

where $\mathbf{v}$ is the object's momentary velocity vector. It is calculated from the trajectory of the center of gravity of f_t.

Waves are generated by changes of f_s, so we use $g_s := \partial f_s/\partial s$ as disturbance. In analogy to equations (2) and (3), g_s acting over the interval $s \in [t, t+\Delta t]$ leads to the following additive change to the elevation field $\eta_{t+\Delta t}(\mathbf{x})$:

$$\Delta\eta_{t+\Delta t}(\mathbf{x}) = \int_t^{t+\Delta t} ds \iint_{\mathbb{R}^2} \frac{d^2\mathbf{k}}{4\pi^2} \iint_{\mathbb{R}^2} d^2\mathbf{y} \times \exp\Big(i(\mathbf{k}\cdot\mathbf{y}-\omega(\mathbf{k})(t+\Delta t-s))\Big)\, g_s(\mathbf{x}-\mathbf{y}).$$

The integral over s can be eliminated by partial integration:

$$\Delta\eta_{t+\Delta t}(\mathbf{x}) = \iint_{\mathbb{R}^2} d^2\mathbf{y}\left(f_t(\mathbf{x}-\mathbf{y})\Phi^{(1)}_{\Delta t,\mathbf{v}}(\mathbf{y}) + f_{t+\Delta t}(\mathbf{x}-\mathbf{y})\Phi^{(2)}_{\Delta t,\mathbf{v}}(\mathbf{y})\right) \tag{10}$$

where

$$\Phi^{(1)}_{\Delta t,\mathbf{v}}(\mathbf{y}) := \iint_{\mathbb{R}^2} \frac{d^2\mathbf{k}}{4\pi^2} \exp\Big(i(\mathbf{k}\cdot\mathbf{y}-\omega(\mathbf{k})\Delta t)\Big) \times \left(-1+i\omega(\mathbf{k})\Delta t\,\frac{e^{-i\phi(\mathbf{k})}+i\phi(\mathbf{k})-1}{\phi(\mathbf{k})^2}\right)$$

and

$$\Phi^{(2)}_{\Delta t,\mathbf{v}}(\mathbf{y}) := \iint_{\mathbb{R}^2} \frac{d^2\mathbf{k}}{4\pi^2} \exp(i\mathbf{k}\cdot\mathbf{y}) \times \left(1+i\omega(\mathbf{k})\Delta t\,\frac{e^{i\phi(\mathbf{k})}-i\phi(\mathbf{k})-1}{\phi(\mathbf{k})^2}\right)$$

with $\phi(\mathbf{k}) := (\mathbf{k}\cdot\mathbf{v}-\omega(\mathbf{k}))\Delta t$.

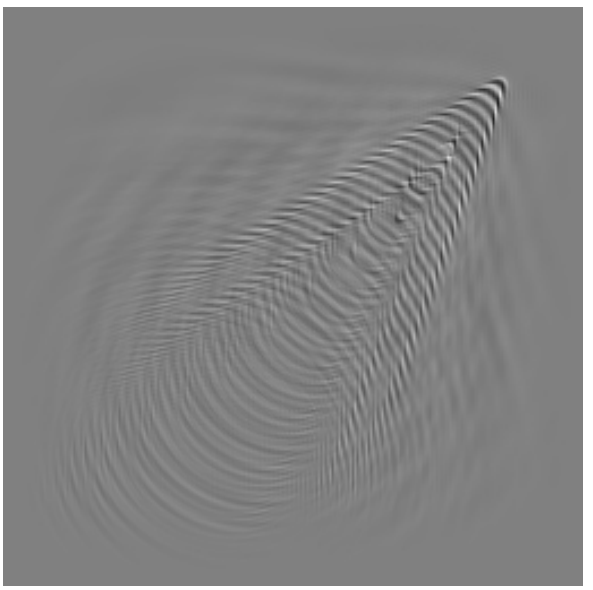

Figure 6: *For ships moving at constant speed in constant direction, the algorithm produces Kelvin wakes, characterized by a wedge with a semiangle of 19°. Parameters: $L = 100\,m$, $\Delta t = 4\,s$, $N = 256$, $2M = 80$.*

Again, we introduce a regularization by modifying the dispersion relation as in equation (7). It helps to subtract an additional small imaginary offset from $\omega(k)$, such as $10^{-20}i$. Otherwise, there could occur division by zero errors or overflow errors when $\phi(\mathbf{k})$ becomes zero or nearly zero. This happens when the phase velocity is approximately equal to the velocity of the object. The subtraction of a small imaginary number to avoid theses problems causes no harm: Actually, the integrands do not possess poles at these values of $\mathbf{k}$ since several contributions cancel.

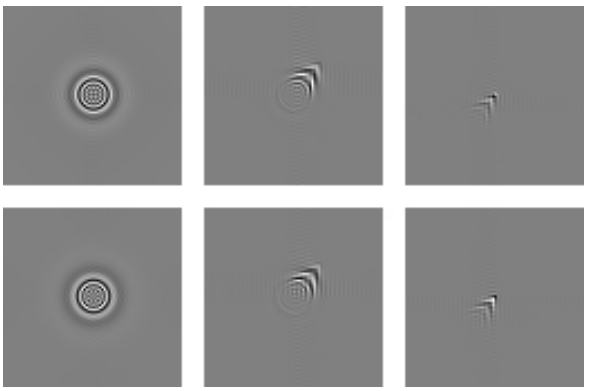

Figure 7: *The convolution kernels leading to the simulation result of figure 6. From left to right:* $\Phi_{\Delta t}$, $\Phi^{(1)}_{\Delta t,\mathbf{v}}$, *and* $\Phi^{(2)}_{\Delta t,\mathbf{v}}$. *Upper row: real part, lower row: imaginary part.*

The resulting expression for $\Delta\eta_{t+\Delta t}(\mathbf{x})$ is discretized in the same way as in equation (4) and cut-off at the border in analogous form to equation (5). It has to be evaluated for every swimming object and for every picture frame except for the first one.

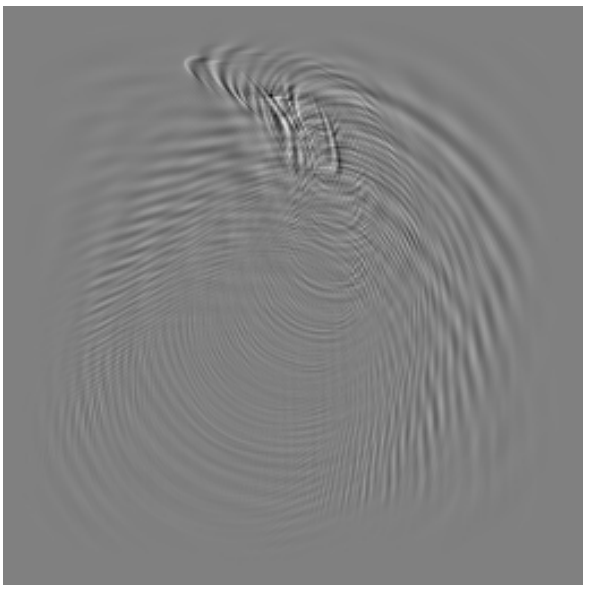

Figure 8: *The algorithm can treat curved ship paths as well; like in nature, the Kelvin wake is distorted. (Simulation parameters as in figure 6. For a 3-D rendering see figure 2.)*

5. Results and Discussion

Figure 1 displays a simulation of raindrops, rendered with Maxon Cinema 4D XL. At pseudo-random moments, the elevation field will be offset at single, pseudo-randomly placed lattice points. Our system reproduces the typical ripples preceding the relatively large inner ring.

The convolution-based approach produces ship wakes which form the correct angle, as can been seen in figure 5. By recomputing the kernels, curved paths can be handled as well, see figure 2. Just like in nature, the wedge pattern of the Kelvin wake is distorted along the curve.

Up to now, the system does neither reflect nor diffract waves at boundaries or swimming objects. Instead, waves move through obstacles. This remains nearly invisible in open-ocean scenes, but can be noticed when shores, islands, or pool walls come into play. In addition, the simulation lacks shallow-water effects, especially the refraction of waves hitting a shore. Non-linear effects are not yet incorporated into the algorithm. Since the water surface is modeled by an elevation field, waves cannot break.

The discretization to a grid causes little visual defects. Of course, the time step Δt and the grid interval Δx must be chosen in such a way that the sliding window contains the important features of a ring wave. The size $2M\Delta x$ of the window must be large enough to capture the major part of the ring wave. On the other hand, the wave must not degenerate to a few pixels in the middle.

To adjust the values of Δt, Δx, and M, one can estimate the size of a ring wave Δt after a point-like disturbance has occurred. This ring has to fit into the window, but should not be much smaller. The sharply defined inner wave front expands radially at a speed set by the minimum $1.1\sqrt[4]{Tg/\rho}$ of the group velocity. The outer front of the ring wave decays much more softly, so that its velocity is harder to define. The asymptotic expansion of gravity waves[2] contains the term $\cos(g(\Delta t)^2/4r - \pi/4)$, where r is the distance from the origin of the disturbance. The largest r for which this expression attains a maximum obeys $g(\Delta t)^2/4r - \pi/4 = 0$. Therefore, an estimate for the outer size of the ring wave can be given as $g(\Delta t)^2/\pi$. In our experiments, we still got convincing results when half of the size of the sliding window was $1/8$ of this size.

This leads to the following conditions on Δt, Δx, and M:

$$\Delta x \overset{!}{\ll} 1.1\sqrt[4]{Tg/\rho}\,\Delta t \overset{!}{\ll} M\Delta x, \quad g(\Delta t)^2/\pi \overset{!}{\approx} 8M\Delta x.$$

Furthermore, the objects' velocity moving across the water must not be too large: The waves they generate within Δt have to fit into the window used for approximating the $\mathbf{k}$-integrals of $\Phi^{(1)}_{\Delta t,\mathbf{v}}$ and $\Phi^{(2)}_{\Delta t,\mathbf{v}}$ in equation 10.

Because there is only one computational step per picture frame, the stability of the solution algorithm is not a very critical issue. In addition, the discretized system obeys a propagation identity corresponding to equation (6) which rules out many run-away effects. An open question remains whether the suppression of boundary reflections according

to equation (5) can cause instability. However, in our experiments such effects were not visible.

Using the prototype software, a PC equipped with an Pentium(R)-III processor running at 500 MHz computes waves at a speed of 90 s per picture frame ($L = 256$, $2M = 80$, one swimming object). The most time-intensive part of the algorithm consists in the convolution, even though we limit the evaluation of equation (10) to the neighborhood of swimming objects and pre-compute $\Phi^{(1)}$ and $\Phi^{(2)}$ per object for each frame. This pre-computation and the discretized calculation of equation 3 could be implemented through a Fast Fourier transform (FFT), but this would not yield an appreciable acceleration.

6. Summary and Future Work

We have presented an approach to simulate water surfaces waves using a simple algorithm. Even though it neglects nonlinearity and viscosity, the model believably reproduces visually important effects such as the distinct behavior of capillary and gravity waves. It produces realistic ring waves and ship waves.

Typically, only a close inspection of the results reveals shortcomings of this algorithm when compared to more complex approaches. To further improve the simulation, one can think of several tasks:

- Simulate ocean waves using a pseudo-random realization of a spectrum measured in nature[18].
- Reflect waves at boundaries; diffract waves at obstacles.
- Generate foam at sharp wave crests. It could be rendered using a texture map or via a particle system[14].
- Simulate wakes stirred by the propellers of ships.
- Calculate velocity data to render motion blurred waves.
- Simulate non-linear effects of hydrodynamics—for instance, by including self-advection[17] or by deforming the waves in order to produce steeper peaks[5].

After freezing the features of the solution, we want to speed up the run-time performance of the algorithm using the floating-point SIMD instructions of common microprocessors to implement convolution and FFT.

References

1. J. X. Chen, N. da Vitoria Lobo. Toward Interactive-Rate Simulation of Fluids with Moving Obstacles Using Navier-Stokes Equations. *Graphical Models and Image Processing*, **57**(2):107–116, 1995. 3
2. L. Debnath. *Nonlinear Water Waves*. Academic Press, 1994. 1, 2, 4, 6
3. N. Foster and D. Metaxas. Controlling Fluid Animation. *Computer Graphics International 97*, pp. 178–188, 1997. 3
4. N. Foster and R. Fedkiw. Practical Animation of Liquids. *SIGGRAPH 2001 Conference Proceedings, Annual Conference Series*, pp. 23–30, 2001. 3
5. A. Fournier and T. Reeves. A Simple Model of Ocean Waves. *ACM Computer Graphics (Proc. of SIGGRAPH '86)*, **20**(4): 75–84, 1986. 3, 7
6. M. N. Gamito and F. K. Musgrave. An Accurate Model of Wave Refraction Over Shallow Water. *Computers & Graphics*, in press. 3
7. J. Ch. Gonzato and B. Le Saëc. On Modelling and Rendering Ocean Scenes—Diffraction, Surface Tracking and Illumination. *Journal of Visualization and Computer Animation*, **11**(1):27–37, 2000. 3
8. M. Gomez. Interactive Simulation of Water Surfaces. *Game Programming Gems*. Edited by M. DeLoura, pp. 187–194, Charles River Media, 2000. 2
9. M. Goss. A Real Time Particle System Display of Ship Wakes. *IEEE Computer Graphics and Applications*, **10**(3):30–35, 1990. 3
10. R. S. Khan *A Simple Model of Ship Wakes*. Master thesis, University of British Columbia, 1994. 3
11. M. Kass and G. Miller. Rapid, Stable Fluid Dynamics for Computer Graphics. *ACM Computer Graphics (Proc. of SIGGRAPH '90)*, **24**(4): 49–57, 1990. 3
12. A. T. Layton and M. van de Panne. A Numerically Efficient and Stable Algorithm for Animating Water Waves. *The Visual Computer*, **18**:41–53, 2002. 3
13. J. F. O'Brien and J. K. Hodgins. Dynamic Simulation of Splashing Fluids. *Proceedings of Computer Animation '95*, pp. 198–205, 1995. 3
14. D. R. Peachey. Modeling Waves and Surf. *ACM Computer Graphics (Proc. of SIGGRAPH '86)*, **20**(4): 65–74, 1986. 3, 7
15. J. Schneider and R. Westermann. Towards Real-Time Visual Simulation of Water Surfaces. *Vision, Modeling, and Visualization 2001*, 211–218, 2001. 3
16. J. Stam. Stable Fluids. *SIGGRAPH 1999 Conference Proceedings, Annual Conference Series*, pp. 121–128, 1999. 3
17. J. Stam. A Simple Fluid Solver Based on the FFT. *Journal of Graphics Tools*, **6**(2):43–52, 2002. 3, 7
18. S. Thon and D. Ghazanfarpour. Ocean Waves Synthesis and Animation Using Real World Information. *Computers & Graphics* **26**:99–108, 2002. 3, 7
19. H. Weimer and J. Warren. Subdivision Schemes for Fluid Flow. *SIGGRAPH 1999 Conference Proceedings, Annual Conference Series*, pp. 111–120, 1999. 3