Complex Water Effects at Interactive Frame Rates

Jörn Loviscach
Hochschule Bremen
Flughafenallee 10
28199 Bremen, Germany
jlovisca@informatik.hs-bremen.de

ABSTRACT

Due to the complexity of the effects, it is difficult to achieve photorealism and physically correct motion in interactive 3-D water simulations. We present a simulation and visualization method which goes a long way towards physical reality, e. g. by producing Kelvin ship waves and displaying images of real caustics on the water surface. Using efficient approximations and employing the specialized 3-D and CPU features of modern PCs, this simulation runs at rates of several frames per second. Part of this solution is the emulation of a hardware accumulation buffer on consumer 3-D graphics cards with the help of texture shaders.

Keywords

water waves, Kelvin waves, caustics, refraction, Fresnel effect, accumulation buffer

1 INTRODUCTION

As an important part of nature, water is an obvious object to simulate in virtual 3-D worlds. Its physical description reads rather simple: a fluid which is almost incompressible, inviscid, and in many cases transparent. However, the faithful reproduction of the dynamic and optical behavior of water requires sophisticated calculations which typically limit interactive water simulations to coarse sketches of the real world.

In this paper, we present an interactive simulation and visualization method for a water surface which reacts to sudden disturbances (such as a stone thrown into a pool) as well as to continuous disturbances such as traveling ships. As can be seen in Fig. 1, the water is rendered with: geometrical deformation by large waves, bump details of small waves, caustics on the ground (seen refracted through the surface), shadows in the caustics, sun highlights, and sky reflection.

To achieve a high frame rate, this method employs special features of today's mainstream PC hardware: first, the render-to-texture capability and the texture shaders of 3-D graphics cards; second, single-instruction/multiple-data computation within the CPU. On a Pentium$^{(R)}$-4 system (2.53 GHz) with a GeForce 4^{TM} Ti 4200 graphics card, a demo scene with a height field resolution of 384×384 runs at approximately 6 fps, virtually independent on the window size.

The main contributions of this work are:

- an interactive simulation of water surface waves producing capillary waves and ship waves which are physically correct in linear approximation,
- a method to generate caustics and shadows within caustics through MIP-mapping,
- a method to emulate a hardware-accelerated accumulation buffer (a feature missing in many 3-D graphics cards) with texture shaders.

Figure 2 sketches the architecture of our prototype implementation with Windows$^{(R)}$ and OpenGL$^{(R)}$. By dragging and clicking with the mouse, the user can drag a ship through the water or place point-like disturbances as if he were dropping stones into a pool.

Section 2 outlines previous work done in the areas of hydrodynamical simulation for 3-D animation purposes and for fast caustics computation. Our wave simulation algorithm is described in section 3. The

Journal of WSCG, Vol. 11, No. 1., ISSN 1213-6972
WSCG '2003, February 3–7, 2003, Plzen, Czech Republic.

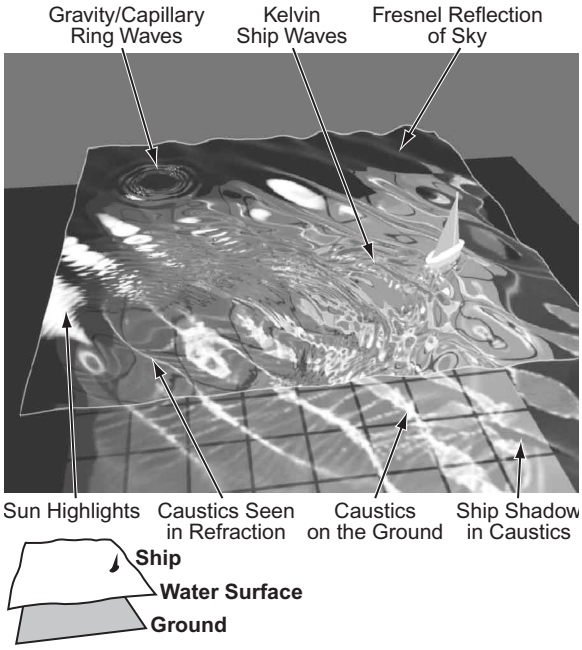

Figure 1. The system simulates the dynamics and optics of water surface waves. (Side length: 15 m)

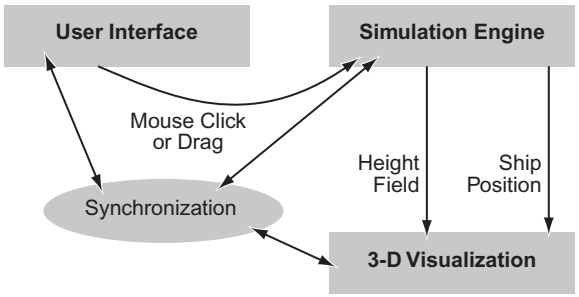

Figure 2. The user interface, simulation, and visualization tasks are assigned to three synchronized processing threads.

basic rendering of the water surface is the topic of section 4, whereas section 5 details how caustics are rendered in our system with 3-D hardware support. For this last step, we require a fast, hardware-based accumulation buffer. Section 6 describes how this can be emulated using consumer 3-D hardware. This solution is interesting in itself, e. g. for image processing applications. The concluding section 7 summarizes the results and offers an outlook on possible future work.

2 PREVIOUS WORK

In a recent survey, Adabala and Manohar [Ada02] identify three major types of fluid models for computer graphics: texture-based (2-D, fractal, or 3-D), combining textures and particles (blobs, grids, or particle maps), and completely particle-based.

The seminal works of Fournier and Reeves [Fou86] as well as Peachey [Pea86] can be seen as forerunners of the 2-D texture-based category. A wide-spread approach to the dynamics of height fields uses finite-difference methods, as described by Kass and Miller [Kas90]. On modern hardware, such an approach easily runs in real time even with a high grid resolution, as, too, does the method of Chen and Lobo [Che95], who replace the full three-dimensional Navier-Stokes equations with a two-dimensional version. Loviscach [Lov02] also uses a height field, but generates time evolution by convolution with a ring wave. This implementation cannot achieve interactive speed but is able to reproduce phenomena like Kelvin ship waves and the complex ring waves formed by gravity and capillary waves on deep water. Glassner [Gla02] geometrically constructs height fields for a simplified model of Kelvin ship waves.

An excellent method to simulate phenomena like breaking waves and poured liquid belongs to the category “textures and particles combined”: Enright, Marschner, and Fedkiw [Enr02] build upon previous work by Foster and Fedkiw [Fos01] as well as Stam [Sta99]. They combine a partial differential equation solver for a voxel-based flow with marker particles to trace the surface. For a grid size of $540 \times 75 \times 120$ the simulation takes 13 minutes per frame.

Trading realism for speed, several researchers have tried to achieve interactive fluid simulations. Schneider and Westermann [Sch01] simulate waves using fractal noise and then render these in real time with light reflection and ground refraction using modern consumer graphics hardware.

Jensen and Goliáš [Jen01] synthesize a height field from a given spectrum and distort it to obtain choppy (Gerstner) waves. Details of the waves are fitted in by the algorithm of Kass and Miller [Kaa90] and by a 2-D version of Stam’s solver [Sta99], all on a 64×64 grid. Light rays going through the vertices of the surface triangles are followed onto the ground, taking into account the refraction according to Snell’s law. These triangles are illuminated inversely proportional to their area, comparable to backward beam tracing [Wat90]. Shafts of light, which are important for underwater scenes, are rendered by slicing the volume into layers which are approximated by blending textured planes.

Hinsinger, Neyret, and Cani [Hin02] also synthesize Gerstner waves. Their scheme adapts to the visible size of each surface element. Thus, it is possible to render views across the water to the horizon at high frame rates.

Several authors discuss the fast rendering of optical effects within water. Nishita and Nakamae [Nis94] cal-

culate so-called illumination volumes formed by each triangle on the water surface and the sun rays entering at its vertices. Scan line by scan line, from top to bottom, this illumination volume is projected into the accumulation buffer. Caustics on curved surfaces in the water are painted at the same time. Illumination volumes hitting objects are cut away for all following (i. e. lower) scan lines, which leads to physically correct shadowing.

Iwasaki, Dobashi, and Nishita [Iwa01] render caustics on the ground like Jensen and Goliáš [Jen01]. Their method paints the above-mentioned illumination volumes as shafts of light by adding filled polygons to the frame buffer. Using additional sample planes together with z- and stencil buffers, caustics can be rendered onto submerged objects in the water. For the generation of shadows, a flat water surface is assumed.

Trendall and Stewart [Tre00] employ several approximations to reformulate the computation of refractive caustics as a convolution problem. High-end graphics hardware is able to handle this at interactive rates.

Research into real-time global illumination is not limited to refractive caustics in water. Jozwowski [Joz02] generates photon maps in real time: For each rendered frame, another 100 photons are sent into the scene, adding to the photons of the former frames. This method obviously cannot cope with the dynamics of a water surface. Sloan, Kautz, and Snyder [Slo02] render global illumination effects for dynamic lighting in real time. However, they mostly rely on pre-computed radiance transfer functions. In addition, their low-frequency sampling is not intended to reproduce complex detail like that of water caustics.

3 SIMULATION OF WAVES

A complete simulation of water waves requires that the velocity field is calculated throughout the entire water volume. This means solving the non-linear Euler equations (i. e. Navier-Stokes equations with zero viscosity) subject to a non-linear boundary condition. [Enr02] goes a long way towards this goal.

Nevertheless, to achieve maximum simulation speed and minute detail, we chose to treat the water surface as a height field. This reduces the computational load by several orders of magnitude so that even surface resolutions of 1024×1024 pixels can be handled interactively. This kind of simulation is limited to ocean or pool waves: Breaking waves, waterfalls etc. cannot be simulated using a height field.

In many approaches, the time development of the height field is calculated with a finite-difference scheme. However, this is only acceptable for the shallow water limit where every wave moves at the same speed, regardless of wavelength. Waves on an ocean of infinite depth, however, can move at an arbitrarily large speed—a phenomenon hard to reproduce with a finite-difference scheme that acts locally. Simulations with fixed wave velocity represent sound waves rather than the surface waves of deep water. Therefore, such simulations lack characteristic effects such as Kelvin ship waves and the complex form of ring waves.

The same is true for Navier-Stokes computations in 2-D: Much of the physics of water waves is determined by the boundary condition at the surface—but this boundary condition is lost when solving only a flattened version of the Navier-Stokes equations.

A basic idea to reproduce typical water wave phenomena without computational overhead is to work in linear approximation and use Huygen's principle: From each point of a wave, a new ring wave emanates. Therefore, to calculate the propagation of the waves in a time span Δt, one convolves the wave field with the profile of a ring wave. This ring wave and thus the convolution kernel Ψ is given [Lov02] by

$$\Psi(\mathbf{x}) = \iint_{\mathbb{R}^2} \frac{d^2\mathbf{k}}{4\pi^2} \exp\big(i(\mathbf{k}\cdot\mathbf{x} - \omega(|\mathbf{k}|)\,\Delta t)\big), \qquad (1)$$

where $\mathbf{x}$ is the 2-D position on the plane of the ocean, $\mathbf{k}$ denotes the wave vector, and the angular frequency $\omega(k)$ fulfills the dispersion law

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

with gravity acceleration $g = 9.8\,\mathrm{kg\,m/s^2}$, surface tension $T = 0.072\,\mathrm{kg/s^2}$, and mass density $\rho = 1.0 \cdot 10^3\,\mathrm{kg/m^3}$ (using typical values).

Sudden disturbances are simply added to the wave field. The waves generated by a continuously moving ship, however, are more difficult to simulate: They must not look as if a series of stones had been dropped into the water. Rather, the waves generated by a ship over one time step of length Δt can be integrated, assuming that a ship follows an approximately linear path with constant velocity $\mathbf{v}$ in this time interval $[t, t+\Delta t]$. Let $f_t(\mathbf{x})$ describe the ship's profile at time t, i. e. this function equals one inside the ship, elsewhere it is zero. Then the waves generated in $[t, t+\Delta t]$ and observed at time $t+\Delta t$ are described using a more complex convolution [Lov02]:

$$\iint_{\mathbb{R}^2} d^2\mathbf{y}\,\big(f_t(\mathbf{x}-\mathbf{y})\Phi_1(\mathbf{y}) + f_{t+\Delta t}(\mathbf{x}-\mathbf{y})\Phi_2(\mathbf{y})\big) \qquad (2)$$

where

$$\Phi_1(\mathbf{y}) := \iint_{\mathbb{R}^2} \frac{d^2\mathbf{k}}{4\pi^2} \exp\big(i(\mathbf{k}\cdot\mathbf{y} - \omega(\mathbf{k})\Delta t)\big) \times \left(-1 + i\omega(\mathbf{k})\Delta t \frac{e^{-i\alpha(\mathbf{k})} + i\alpha(\mathbf{k}) - 1}{\alpha(\mathbf{k})^2}\right),$$

$$\Phi_2(\mathbf{y}) := \iint_{\mathbb{R}^2} \frac{d^2\mathbf{k}}{4\pi^2} \exp(i\mathbf{k}\cdot\mathbf{y}) \times \left(1 + i\omega(\mathbf{k})\Delta t \frac{e^{i\alpha(\mathbf{k})} - i\alpha(\mathbf{k}) - 1}{\alpha(\mathbf{k})^2}\right)$$

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

The simulation then runs as an infinite loop:

- Propagate by convolving with expression (1).
- Add new sudden disturbances, if any.
- Add new ship waves according to expression (2).
- Cut off the waves at boundaries (see below).
- Repeat.

For computation, the wave field is discretized to a grid of size $N \times N$ with e. g. $N = 512$. The new ship waves calculated in the third step only are of importance in the neighborhood of the ship. This is covered by a smaller domain, e. g. 64×64. Call this $2M \times 2M$. Of course the time step Δt must be small enough to prevent the waves from visibly leaking past this area.

To simulate an infinitely extended ocean on the finite square grid $N \times N$, we have to get rid of boundary effects. As in [Lov02], a margin of M (for M see preceding paragraph) is used, so that only the inner $(N-2M) \times (N-2M)$ points of the $N \times N$ grid appear in the visualization. However, where [Lov02] used a linear cutoff, a quadratic cutoff turned out to be better and effective in all experiments. This means that in the fifth step, the wave field at grid position (p,q) with $0 \le p,q < N$ is multiplied by $h(p)\,h(q)$ defined by

$$h(p) := \begin{cases} 1 - \left(\frac{p}{M} - 1\right)^2 & \text{if } p < M \\ 1 & \text{if } M \le p < N - M \\ 1 - \left(\frac{N-1-p}{M} - 1\right)^2 & \text{if } p \ge N - M. \end{cases}$$

The time-limiting step in the calculation is the convolution on the $N \times N$ grid. The computation of the ship waves on the much smaller domain $2M \times 2M$ is negligible in comparison. To obtain optimum speed, all convolutions are done with Fast Fourier Transform (FFT), multiplication with the pre-calculated Fourier transform of the kernel, and inverse FFT. The cutoff by $h(p)\,h(q)$ prevents problems with the periodicity of the Fourier transform.

In the Fourier domain it is easy to add a pseudo-random spectrum of wind waves, as can be seen in Figure 1. Furthermore, in the Fourier domain we can blur the profile of the ship by multiplying by a Gaussian. The resulting wave spectrum contains fewer small wavelengths, simulating the impression of real ship waves.

In the prototype, the FFT and most other parallelizable operations are implemented with the aid of Intel$^{(R)}$'s C function library called Integrated Performance Primitives 2.0 (IPP). This library provides arithmetic and image-processing functions based on the single-instruction/multiple data features of modern PC processors, known as MMX^{TM} and Streaming SIMD Extensions (SSE).

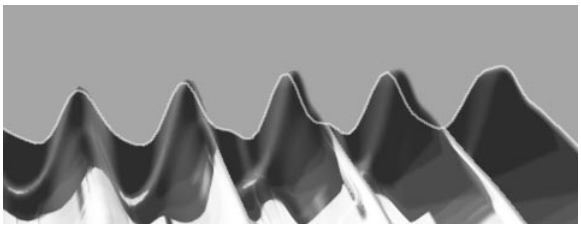

Figure 3. Second-order effects lead to steep wave crests and flat troughs. (Image detail)

In [Lov02], the wave field produced by the convolution computations is directly used as a height field η; here this wave field is used as velocity potential ϕ. This allows to take into account a certain non-linear effect up to the second order: The stronger a wave, the steeper are its crests and the flatter its troughs, see Figure 3. Along the line of analysis of Stokes waves given in [Deb94] one can approximate this effect as follows: If the velocity potential is sinusoidal

$$\phi(\mathbf{x},t) = a\,e^{i(\mathbf{k}\cdot\mathbf{x} - \omega t)}$$

with a fixed amplitude a, wave vector $\mathbf{k}$ and angular frequency $\omega = \omega(|\mathbf{k}|)$ then the height field up to second order in a is

$$\eta(\mathbf{x},t) = ai\frac{|\mathbf{k}|}{\omega} e^{i(\mathbf{k}\cdot\mathbf{x} - \omega t)} - a^2 \frac{|\mathbf{k}|^3}{2\omega^2} e^{2i(\mathbf{k}\cdot\mathbf{x} - \omega t)}. \quad (3)$$

For arbitrary ϕ, a corresponding calculation can easily be done in the Fourier domain. This means that for each full simulation step, three expensive $N \times N$ FFT operations to be done: twice a complex-to-complex FFT (forward/inverse) for the convolution and one complex-to-real FFT to calculate the height field.

We have found a way to triple the perceived speed of the algorithm: Since the method is stable for large time steps, the time step can be increased by a factor of 3. For each such step $\eta(\mathbf{x}, t + \Delta t)$ is computed;

both $\eta(\mathbf{x},t+\frac{1}{3}\Delta t)$ and $\eta(\mathbf{x},t+\frac{2}{3}\Delta t)$ can be interpolated. This interpolation cannot be linear due to the strongly oscillating nature of the waves, but a cubic approximation turns out to be effective:

$$\begin{aligned}\eta(\mathbf{x},t+\frac{1}{3}\Delta t) \quad &\approx \quad \frac{20}{27}\eta(\mathbf{x},t)+\frac{7}{27}\eta(\mathbf{x},t+\Delta t)\\ &\quad +\frac{4}{27}\dot{\eta}(\mathbf{x},t)-\frac{2}{27}\dot{\eta}(\mathbf{x},t+\Delta t),\end{aligned}$$

and similarly for $\eta(\mathbf{x},t+\frac{2}{3}\Delta t)$. Here, $\dot{\eta}$ is the time derivative of η. It can be calculated from ϕ in analog manner to equation (3).

Because only the real parts of η and $\dot{\eta}$ are of interest, both can be determined from ϕ in a single intertwined $N \times N$ complex-to-complex FFT which computes $\mathrm{Re}(\eta)+i\mathrm{Re}(\dot{\eta})$. Therefore there are three $N \times N$ FFT operations per *full* simulation step, but effectively there is only one such operation per displayed frame.

4 RENDERING THE SURFACE

The surface of clear water mainly becomes visible through reflection and refraction. Our method calculates the reflections of the sun and of a uniformly blue sky as well as the refraction of the ground together with the caustics and the ship's shadow within them (see section 5).

Using a simulation of 512×512 grid points with an invisible, 64 points wide border results in a displayed height field of 384×384 grid points. The number of pixels on the screen roughly corresponds to this number. Drawing the height field as 383×383 quadrangles would therefore not be efficient.

A fast method to render the water surface would be to use a single large quadrangle with an appropriate texture. However, the missing of geometry would be obvious when rendering high waves or ship walls. Hence, we chose a compromise similar to [Jen01, Lov02]: The surface is only deformed by waves of a large wavelength, to which the effect of the small wavelengths is added by perturbing the normals.

For a good balance between rendering speed on one hand and detail in geometry on the other, a subsampling factor of 4 proved to be appropriate. Therefore, the water surface consists e. g. of 95×95 instead of 383×383 quadrangles, each of which is rendered as two triangles. The height field of the reduced geometry is found by weighted average.

Through multitexturing, several textures are applied to this surface to hide its rough geometry. The main texture is identical to the ground texture (see section 5), except that it is applied as a refraction map, i. e. with distorted coefficients. These originate from two sources. First, each of the 95×95 vertices has texture coordinates which describe which point of the ground would be visible here if the water was flat and located at the height of this vertex. This is calculated through linear approximation. Figure 4 shows the effectiveness of this approach: If the refraction index of the water is set to 1, the considerable geometric deformation is undone almost perfectly.

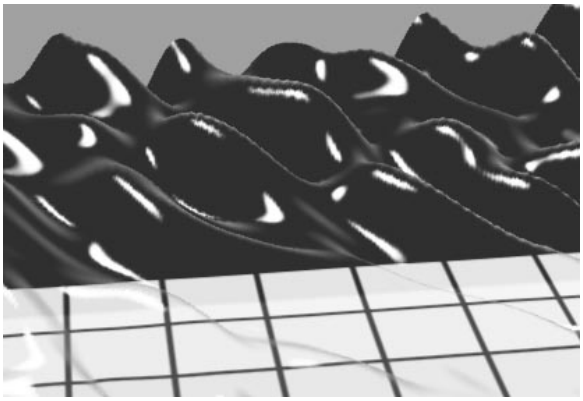

Figure 4. At a refraction index of 1, the effectiveness of the linear approximation of the texture coordinates becomes obvious: The tile pattern appears almost without distortion. (Image detail)

The "offset texture 2-D" mode of texture shaders allows to displace these coarse texture coordinates pixel by pixel according to another texture. For this we use an image of size e. g. 512×512 which contains a linear approximation of the effect of the refraction due to the inclination of the normals.

In total, the texture coordinates of the refraction map are calculated as follows:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \underbrace{\begin{pmatrix} u_0 \\ v_0 \end{pmatrix} + \overline{\eta}\begin{pmatrix} a \\ b \end{pmatrix}}_{\substack{\text{given per vertex,} \\ \text{lin. interpolation}}} + \underbrace{\begin{pmatrix} c & d \\ e & f \end{pmatrix}\begin{pmatrix} \eta_u \\ \eta_v \end{pmatrix}}_{\substack{\text{given per} \\ \text{original grid point}}}$$

Here, $\begin{pmatrix} u_0 \\ v_0 \end{pmatrix}$ is the texture coordinate with the water at rest, $\overline{\eta}$ is the local mean value of η, also used for geometry, and η_u, η_v are the differences of the height field along the indicated direction. The spatially varying values $a, \ldots, f$ are found on initialization by taking numerical derivatives of an exact geometric calculation based on Snell's law of refraction.

Two additional high-resolution textures are used to add reflections to the water surface. Both the normalized vector **v** and the normalized half vector **h** between the viewer and the sun are computed at initialization. This is done for every visible point of the original grid.

Then, during visualization, for every frame the normal vector **n** is calculated at each of these points as well. We have found that the following expressions give fast and faithful approximations to Fresnel sky reflection and glaring sun highlights, respectively:

$$\begin{aligned} f &:= \text{clamp}(1.0 - 2.5\,\mathbf{n}\cdot\mathbf{v}), \\ g &:= \text{clamp}(100.0\,\mathbf{n}\cdot\mathbf{h} - 98.95), \end{aligned}$$

where the clamp restricts the values to $[0,1]$. Like almost all other operations, these calculations are done almost exclusively with help of the IPP function library. The results f and g are used as the alpha channel of a blue texture (sky reflection) or as luminance texture (sun highlights), respectively.

A further speed-up could result by offloading the field of normal vectors as texture to the graphics card and using more sophisticated texture shaders such as "Dot Product Reflect Cube Map" and texture combiners. However, since many graphic chips can access only four textures simultaneously, the complete rendering would then typically demand multiple passes.

5 PAINTING CAUSTICS

The system uses a subsampled version of the height field (e. g. 96×96 grid points) to generate caustics and render them into a texture. This texture is used on the ground (a large square) and—with distorted texture coordinates—on the water surface, see section 4.

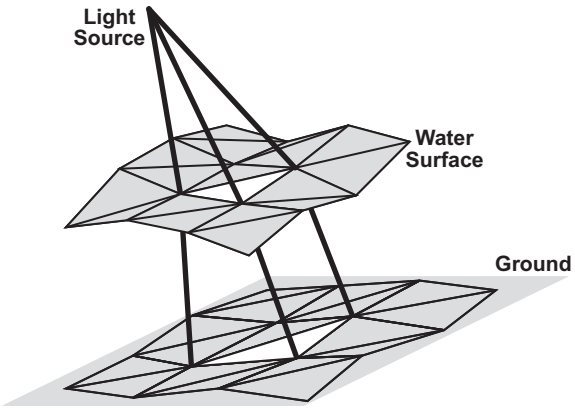

Figure 5. The caustics simulation computes light rays through each vertex by linear approximation.

The caustics computation proceeds as in [Wat90]: Light rays are sent from the sun through each vertex on the water and refracted according to the normal vector. Thus, each triangle on the surface is projected down to a distorted triangle on the floor, see Figure 5. The calculation of the 2-D coordinates (x,y) on the floor can be done by linear approximation:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \overline{\eta}_x \\ \overline{\eta}_y \end{pmatrix}$$

Here, $\overline{\eta}_x$, $\overline{\eta}_y$ are the differences of the height field along the indicated direction. The spatially varying values x_0, y_0, a, …, d are computed during initialization like the (different) coefficients in section 4.

Due to the distortion, the triangles on the ground partially overlap each other. This means that the respective shafts of light are overlaid on the floor—an effect which is simple to achieve with additive blending.

Every triangle on the ground receives the same *total* amount of light, i. e. its luminance has to be inversely proportional to its area. This can be taken into account by giving each triangle a corresponding gray tone, as in [Jen01]. However, in order to save CPU time, we offload the area calculation to the graphics card—through MIP (multum in parvo) mapping. Normally, MIP mapping is applied to diminish aliasing due to texture pixels being smaller than screen pixels: The graphics chip automatically uses a reduced version of the texture with a suitable resolution. To this end, different detail levels of the texture have to be prepared as levels of a MIP map. In our application however, we simply fill the MIP map levels with constant-color images, each level four times brighter than the previous one, until white is reached. As the prototype shows, the hardware interpolation between MIP map levels hides these steps very well.

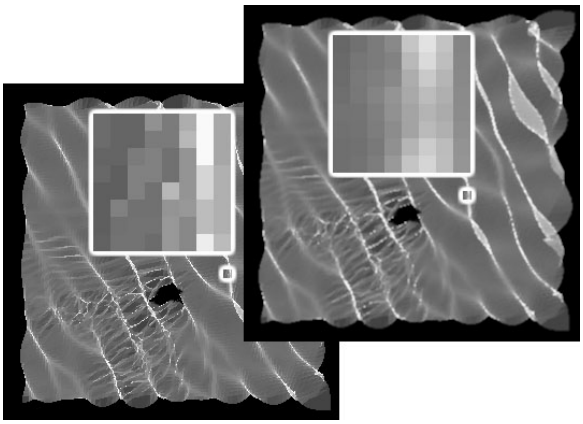

Figure 6. The rendered caustics image is blurred to hide triangles and aliasing. (Enlarged sections framed, Colors adjusted for print)

Thus, the caustics can be rendered as an OpenGL(R) vertex array, given that an appropriate blending mode and MIP map texture are chosen. However, since each triangle is rendered in a single color, often different from its neighbors, the edges of the triangles become visible. In addition, many graphics cards do not support fullscreen antialiasing in offscreen graphics buffers. To fix these problems, the resulting texture image can be blurred as shown in Figure 6, see section 6.

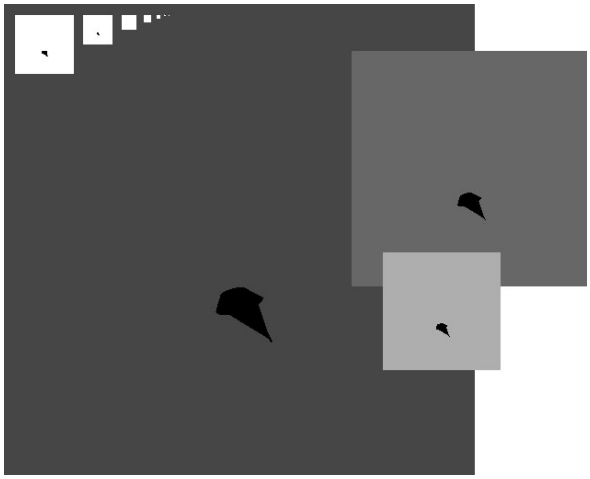

Figure 7. The ship's shadow on the water surface is rendered into the lower levels of the MIP map used for caustics. (Colors adjusted for print)

To take the shadow of a ship into account, we project it onto the water surface using a matrix transformation in OpenGL(R). The result, i. e. the shadow on the surface, is rendered in black into the MIP map, see Figure 7. (The higher levels are skipped because of their low spatial resolution.) After rendering this texture on the deformed vertices, the shadow is automatically placed on the floor at the proper position.

6 ACCUMULATION BUFFER

The caustics generation following section 5 results in an texture image that has yet to be blurred. A typical approach in OpenGL(R) would be to use convolution or to write multiple, shifted renderings into an accumulation buffer. However, many consumer-level graphics cards do not support these functions in hardware. In addition, reading from a graphics buffer takes up excessive time, so that the PC processor cannot help much. Using texture shaders one can emulate a single-color accumulation buffer with 12 bit depth. This is sufficient to average 16 gray scale images without a loss in precision. The way texture shaders and shifted textures are applied here partially resembles the lattice physics by Harris e. a. [Har02].

When applied to blurring a given 8 bit texture, the method runs as follows, see Figure 8:

- Switch to a RGBA graphics buffer with the same dimensions and a depth of 4×8 bits. Its green channel will later contain the sum of the higher 4 bits, its blue channel the sum of the lower 4 bits.
- Configure the texture to be blurred for dependent texture lookup with help of a precalculated texture: If the original texture contains a in its higher four bits and b in its lower four bits, the resulting texture has a in the lower four bits of the green channel, b in the lower four bits of the blue channel. The remaining bits of these channels are set to zero.

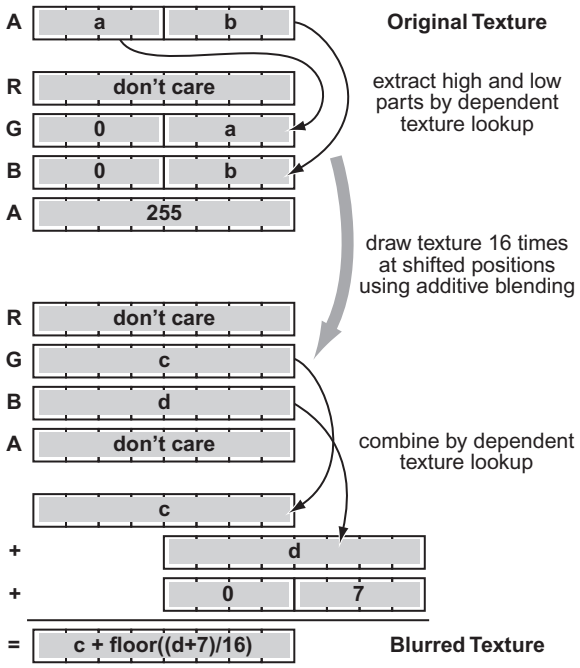

Figure 8. For blurring a gray scale texture, the high and low four-bit-parts of its values are treated separately with the help of texture shaders.

- Activate additive blending of the texture.
- Render it 16 times: 4 times at its original position, 2 times with a shift of one pixel to the left etc. to create a convolution with $\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$, an approximation to Gaussian blur.
- Switch to another graphics buffer of the same dimensions and a depth of 8 bits. It will later contain the blurred texture as a gray scale image.
- Configure the graphics buffer used in the previous steps as a texture with dependent texture lookup. Use a precalculated texture to the following effect: If the previous graphics buffer contains c in the green channel and d in the blue channel, write a gray level of $c + \lfloor (d+7)/16 \rfloor$.
- Use the content of the new buffer as texture.

7 RESULTS. OUTLOOK

We have described an simulation and visualization

Computation Step	Time
wave propagation ($\frac{1}{3}$ of full step)	52 ms
generate ship waves ($\frac{1}{3}$ of full step)	6 ms
generate and blur caustics	18 ms
prepare sun and sky reflection textures	51 ms
render water surface	42 ms

Table 1. Time spent per rendered image in the main steps. See text for details on the system.

system that simulates and renders a water surface with many of its complex hydrodynamic and optical features. The prototype works at an interactive frame rate: It achieves 5.9 fps on a Pentium(R)-4 system (2.53 GHz) under Windows(R) XP with a GeForce 4TM Ti 4200 graphics card, a total grid size of $N = 512$, a margin of $M = 64$ and an original-grid-to-vertex subsampling factor of 4.

Even though the calculation of the ship waves and the caustics proceeds relatively fast, three steps of the method limit the speed: the $N \times N$ FFT used for propagation, the precalculation for sun and sky reflections, and the drawing of the water surface, see Table 1. Obviously, with the FFT and the drawing being hard to accelerate, the reflection calculation presents itself as a starting point for further improvement. In addition, the system could be adapted for underwater scenes with shafts of light rendered using the MIP-mapping method introduced for caustics.

While the presented hydrodynamical simulation is sufficient for open-ocean scenes with small ships, it can still be extended. Especially, we look for ways to incorporate water wave reflections at boundaries—which notably includes ship walls, an effect neglected in the presented simulation. Further topics include the refraction of water waves over depth variations in the ground (e. g. in front of a shore), the simulation of which has been studied thoroughly starting with [Fou86, Pea86].

Further investigation can be spent on the diffraction at barriers and how to model additional effects caused by the non-linearity of the hydrodynamical equations. If the elevation and wavelength are of comparable size, the linear approximation can hardly be justified mathematically while still delivering plausible-looking simulations. However, additional means such as a power series expansion—as partially applied in section 3—might slightly improve the results.

8 REFERENCES

[Ada02] Adabala, N., Manohar, S. Techniques for Realistic Visualization of Fluids: A Survey. Computer Graphics Forum 21, No. 1, pp. 65–81, 2002.

[Che95] Chen, J. X., da Vitoria Lobo, N. Toward Interactive-Rate Simulation of Fluids with Moving Obstacles Using Navier-Stokes Equations. Graphical Models and Image Processing, Vol. 57, No. 2, pp. 107–116, 1995.

[Deb94] Debnath, L. Nonlinear Water Waves. Academic Press, 1994.

[Enr02] Enright, D., Marschner, St., Fedkiw, R. Animation and Rendering of Complex Water Surfaces. Proc. of SIGGRAPH '02, pp. 736–744, 2002.

[Fos01] Foster, N., Fedkiw, R. Practical Animation of Liquids. Proc. of SIGGRAPH '01, pp. 23–30, 2001.

[Fou86] Fournier, A., Reeves, T. A Simple Model of Ocean Waves. Proc. of SIGGRAPH '86, Vol. 20, Nr. 4, pp. 75–84, 1986.

[Gla02] Glassner, A. Andrew Glassner's Notebook: Duck! IEEE Computer Graphics and Applications 22, No. 4, pp. 88–97, 2002.

[Har02] Harris, M. J., Coombe, G., Scheuermann, Th., Lastra, A. Physically-Based Visual Simulation on Graphics Hardware. SIGGRAPH/Eurographics Workshop on Graphics Hardware, pp. 109–118, 2002.

[Hin02] Hinsinger, D., Neyret, F., Cani, M.-P. Interactive Animation of Ocean Waves. Proceedings of ACM SIGGRAPH Symposium on Computer Animation, pp. 161–166, 2002.

[Jen01] Jensen, L. S., Goliáš, R. Deep Water Animation and Rendering. presented at Game Developer's Conference Europe 2001. www.gamasutra.com/gdce/jensen/jensen_01.htm, 2001.

[Iwa01] Iwasaki, K., Dobashi, Y., Nishita, T. Efficient Rendering of Optical Effects within Water Using Graphics Hardware. Pacific Graphics 2001, pp. 374–383, 2001.

[Joz02] Jozwowski, T. R. Real Time Photon Mapping. Master's thesis, Michigan Technological University, 2002

[Lov02] Loviscach, J. A Convolution-Based Algorithm for Animated Water Waves. Eurographics 2002 Short Paper Presentations, pp. 381–389, 2002.

[Nis94] Nishita, T., Nakamae, E. Method of Displaying Optical Effects within Water using Accumulation Buffer. Proc. of SIGGRAPH '94, pp. 373–379, 1994.

[Pea86] Peachey, D. R. Modeling Waves and Surf. Proc. of SIGGRAPH '86, Vol. 20, No. 4, pp. 65–74, 1986.

[Kas90] Kass, M., Miller, G. Rapid, Stable Fluid Dynamics for Computer Graphics. Proc. of SIGGRAPH '90, Vol. 24, No. 4, pp. 49–57, 1990.

[Sch01] J. Schneider and R. Westermann. Towards Real-Time Visual Simulation of Water Surfaces. Vision, Modeling, and Visualization 2001, pp. 211–218, 2001.

[Slo02] Sloan, P.-P., Kautz, J., Snyder, J. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. Proc. of SIGGRAPH '02, pp. 527–536, 2002.

[Sta99] Stam, J. Stable Fluids. Proc. of SIGGRAPH '99, pp. 121–128, 1999.

[Tre00] Trendall, Ch., Stewart, A. J. General calculations using graphics hardware, with application to interactive caustics. Proceedings of Eurographics Workshop on Rendering 2000, pp. 287–298, 2000.

[Wat90] Watt, M. Light-Water Interaction Using Backward Beam Tracing. Proceedings of SIGGRAPH '90, pp. 377–385, 1990.