Lattice-Boltzmann Water Waves

Author One, Author Two, Author Three, Author Four

Authors' Affiliations

Abstract. A model for real-time generation of deep-water waves is suggested. It is based on a lattice-Boltzmann (LB) technique. Computation of wave dynamics and (ray-traced) rendering for a lattice of size 1024^2 can be carried out simultaneously on a single graphics card at 25 frames per second. In addition to the computational speed, the LB technique is seen to offer a simple and physically accurate method for handling both dispersion and wave reflection from obstructing objects.

1 Introduction

The goal of this effort is to provide the mathematical basis for a particularly simple, real-time computational model of deep-water waves. Computation of wave dynamics and a ray-traced rendering of the wave height field can be carried out simultaneously, in real-time, on a single NVIDIA GTX 480 graphics card.

The model is based on a lattice-Boltzmann method. Lattice-Boltzmann (LB) methods are a class of *cellular automata* (CA), a collection of computational structures that can trace their origins to John Conway's famous *Game of Life* [1], which models population changes in a hypothetical society that is geographically located on a rectangular lattice. In Conway's game, each lattice site is labeled as populated or not, and each lattice site follows only local rules, based on nearest-neighbor populations, in synchronously updating itself as populated or not. Although the rules are only local, global behavior emerges in the form of both steady-state population colonies and migrating colonies who can generate new steady-state colonies or destroy existing ones.

In a general CA, arbitrary graphs and local rules for vertex updates may be postulated, but those that are most interesting exhibit a global behavior that has some provable characteristic. Lattice-Boltzmann methods employ synchronous, neighbor-only update rules on a discrete lattice, but the discrete populations at each lattice point have been replaced by continuous distributions of some quantity of interest. The result is that the provable characteristic is often quite powerful: the system is seen to converge, as lattice spacing and time step approach zero, to a solution of a targeted class of partial differential equations (PDEs).

Lattice-Boltzmann methods are thus often regarded as computational alternatives to finite-element methods (FEMs), and as such they have have provided significant successes in modeling fluid flows and associated transport phenomena [2–6]. They provide stability, accuracy, and computational efficiency comparable to FEMs, but they offer significant advantages in ease of implementation, parallelization, and an ability to handle interfacial dynamics and complex boundaries. The principal drawback to the methods, compared to FEMs, is the counter-intuitive direction of the derivation they require. Differential equations describing the macroscopic system behavior are derived (emerge) from a postulated computational update (the local rules), rather than the reverse.

The paper is organized as follows. After discussing related work in the next section, we describe our computational model (the local rules) in Section 3. In Section 4, we derive the wave equation directly from the postulated local rules. Section 5 contains a brief discussion of dispersion and wave number spectra, and Section 6 describes initial conditions. A principal benefit of our approach is the ease with which we can handle wave reflections, and we describe this in Section 7. Finally, we provide implementation details in Section 8 and conclusions in Section 9.

2 Related Work

The graphics literature on physically-based modeling and rendering of water flow is extensive. Foundational work by Kass and Miller [7], Foster and Metaxas [8], Foster and Fedkiw [9], and Stam [10], among others, has led to numerous, visually stunning examples of water flow on small to medium scale, such as water pouring into a glass or water sloshing in a swimming pool. Large-scale, deep-water simulations appropriate for oceans or lakes, which is our focus here, usually avoid full-scale, 3D Navier-Stokes solutions and instead employ 2D spectral approaches to simulate displacement of the free surface. Mastin et al. [11] was probably the first. In this case weights in frequency space are obtained by sampling from models fitted to observed spectra, e.g., Hasselmann et al. [12], and then applying a fast Fourier transform to construct the height field. As seen in the work of Jensen et al. [13], this approach can offer real-time performance suitable for interactive gaming [14]. The principal drawback to FFT-based approaches is their inability to handle obstructions, i.e., wave/object interactions. Hinsinger et al. [15] also achieve visually impressive results in real-time using an adaptive sampling, procedural approach that includes dispersion, but again, they do not consider wave obstructions.

In a somewhat complementary approach, Yuksel, House, and Keyser [16] focus on obstructions. They use *wave particles*, which are dynamically blended cosine segments, to provide extremely effective, real-time wave-object interaction. They do not include dispersion, and so their technique is not appropriate for deep-water waves. Nevertheless, they provide impressive demonstrations of large-scale, open water simulations by augmenting their technique with those of Tessendorf [17, 18].

Applications of lattice-Boltzmann methods to water flow are also numerous. Salmon [19] provided an early application to ocean circulation modeling, in particular, a "reduced-gravity" model in which a homogeneous, wind-driven layer of fluid overlays a denser layer that remains at rest. More recently, Thürey et al. [5] use a full, multi-phase 3D LB model to create impressive animations that include object/free-surface interaction for open water. As yet this approach is computationally-intensive, even for relative small grids. They report 80 seconds per frame for a 120^3 grid.

Our approach is most closely related to the *iWave* system of Tessendorf [18], in that we apply 2D site updates based on local information. Ours is a collision matrix, whereas *iWave* uses a convolution kernel applied over a 13×13 or larger neighborhood. The total computational effort is remarkably similar. The advantage of our approach is the flexibility it offers in handling wave-object interaction. *iWave* uses a grid bit mask to indicate object position, and its update operation simply forces wave height to zero at masked sites. A fortuitous consequence of its kernel, which controls damping of the second derivative of height, is that zeroing the height removes this damping and yields a visually effective simulation of reflection. The additional flexibility we offer includes selected directional reflection, damping energy in response to varying restitutional characteristics of the obstruction, changing wave numbers for harmonics, and simulating semi-porous surfaces.

3 The Computational Model

Although 3D grids are common in LB models, we seek to achieve real-time performance, and so we restrict our development to a 2D, rectangular grid with four, unit-length directions, $\boldsymbol{c_i}$, $i = 1, ..., 4$, and a single zero-length direction, $\boldsymbol{c_0}$, as shown in Figure 1. Although 2D, rectangular grids can generate anisotropic

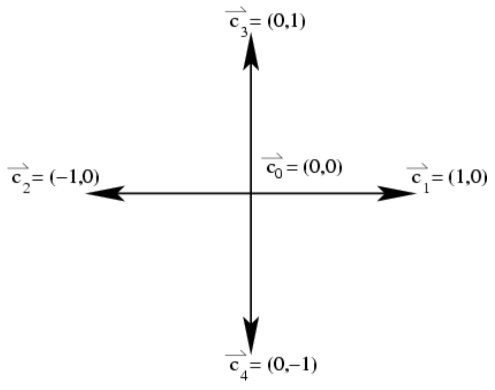

Fig. 1. Model grid.

flows for certain LB models, we will see that anisotropy is avoided here through a careful choice of the site collision matrix.

We assume a lattice spacing, λ, a time step, τ, unit velocity $v = (\lambda/\tau)$, and velocity vectors $\boldsymbol{v_i} = v\boldsymbol{c_i}$, $i = 0, ..., 4$. We further assume that $h(\boldsymbol{r}, t)$, the wave height at site $\boldsymbol{r}$ and time t, comprises 5 directional flows,

$$h(\boldsymbol{r},t) = \sum_{i=0}^{4} f_i(\boldsymbol{r},t) \tag{1}$$

where $f_i(\boldsymbol{r},t)$, represents the mass flow at location $\boldsymbol{r}$ at time t moving in direction $\boldsymbol{c_i}$. The velocity field is then

$$\boldsymbol{u}(\boldsymbol{r},t) = (\sum_{i=0}^{4} \boldsymbol{v_i} f_i(\boldsymbol{r},t))/(\sum_{i=0}^{4} f_i(\boldsymbol{r},t))$$

and the momentum tensor is

$$\Pi_{\alpha\beta} = \sum_{i=0}^{4} v_{i\alpha} v_{i\beta} f_i(\boldsymbol{r},t),$$

where $\alpha, \beta \in$\{x,y\}. Note that for the limited set of directions we use, $v_{i\alpha}v_{i\beta} = v^2\delta_{\alpha\beta}$, and so Π is diagonal.

The fundamental system update equation (basis for simulation) is given by:

$$f_i(\boldsymbol{r} + \lambda\boldsymbol{c_i}, t+\tau) = f_i(\boldsymbol{r},t) + \Omega_i \cdot f(\boldsymbol{r},t), \qquad \text{i = 0,1, ..., 4} \tag{2}$$

where Ω_i is the i^{th} row of a matrix $\Omega : \Re^5 \to \Re^5$, which is a *collision matrix* in the sense that $\Omega_{i,j}$ represents the deflection of flow f_j into the i^{th} direction. Once Ω is specified, equation (2) is, essentially, the entire computational model. Starting with initial conditions, we apply (2) synchronously to all lattice sites and then generate the new wave height field at time $t = t + \tau$ by (1).

The choice of Ω determines the properties of the system. Some important constraints on this choice can be specified immediately. From (2) we have:

- conservation of mass: $\sum_{i=0}^{4} \Omega_i \cdot f(\boldsymbol{r},t) = 0$
- conservation of momentum: $\sum_{i=0}^{4} \boldsymbol{v_i}\Omega_i \cdot f(\boldsymbol{r},t) = (0,0)$

The principal constraint is that the limiting behavior of (2) as $\lambda, \tau \to 0$ should be a recognizable wave equation.

We choose to specify

$$\Omega = \begin{pmatrix} -4K & 2-4K & 2-4K & 2-4K & 2-4K \\ K & K-1 & K-1 & K & K \\ K & K-1 & K-1 & K & K \\ K & K & K & K-1 & K-1 \\ K & K & K & K-1 & K-1 \end{pmatrix} \tag{3}$$

where $K \in [1/4, 1/2]$ is a parameter. We will see that this choice ultimately yields a limiting wave equation with speed (phase velocity) $v\sqrt{K}$. For now we note that 0 is a triple eigenvalue of Ω and that the eigenvectors $e_0 = (2-4K, K, K, K, K)$, $e_1 = (0,1,-1,0,0)$, and $e_2 = (0,0,0,1,-1)$ span the null space.

4 Derivation of the Wave Equation

In this section, we show that the limiting behavior of (2) as $\lambda, \tau \to 0$ is indeed the well-known wave equation. Intermediate results include the continuity equation, which is a statement of conservation of mass, and the Euler equation of hydrodynamics, which is a statement of conservation of momentum. It should be noted that, although this derivation is essential in verifying that our model is physically correct, model implementation does not depend upon the derivation in any way.

4.1 The Continuity Equation

We begin with a standard Chapman-Enskog expansion [2]. If we apply a Taylor expansion to the basic update equation (2) we obtain:

$$[(\lambda \boldsymbol{c_i}, \tau) \cdot \nabla] f_i(\boldsymbol{r}, t) + \frac{[(\lambda \boldsymbol{c_i}, \tau) \cdot \nabla]^2}{2!} f_i(\boldsymbol{r}, t) + ... = \Omega_i \cdot f(\boldsymbol{r}, t) \tag{4}$$

As noted, we want to consider the limiting behavior here as $\lambda, \tau \to 0$; they can, of course, approach at different rates, but we assume they do not. Specifically, we write

$$t = \frac{s}{\epsilon} \qquad \text{where} \quad s = o(\epsilon)$$

$$\boldsymbol{r} = \frac{\boldsymbol{q}}{\epsilon} \qquad \text{where} \quad \boldsymbol{q} = o(\epsilon)$$

and where the limit of interest is $\epsilon \to 0$. Then

$$\frac{\partial}{\partial t} = \epsilon \frac{\partial}{\partial s}$$

$$\frac{\partial}{\partial r_\alpha} = \epsilon \frac{\partial}{\partial q_\alpha} \qquad \text{for} \qquad \alpha \in \{x, y\}$$

So

$$\nabla = (\partial/\partial r_x, \partial/\partial r_y, \partial/\partial t) = \epsilon(\partial/\partial q_x, \partial/\partial q_y, \partial/\partial s) \tag{5}$$

We also assume that the solution, $f(\boldsymbol{r}, t)$, is a small perturbation on this same scale about some local equilibrium, i.e.,

$$f(\boldsymbol{r}, t) = f^0(\boldsymbol{r}, t) + \epsilon f^1(\boldsymbol{r}, t) + \epsilon^2 f^2(\boldsymbol{r}, t) + ... \tag{6}$$

To qualify as a local equilibrium, f^0 must carry the macroscopic quantities of interest, that is,

$$h(\boldsymbol{r}, t) = \sum_{i=0}^{4} f_i^0(\boldsymbol{r}, t) \tag{7}$$

and

$$\boldsymbol{u}(\boldsymbol{r}, t) = (\sum_{i=0}^{4} \boldsymbol{v_i} f_i^0(\boldsymbol{r}, t))/(\sum_{i=0}^{4} f_i^0(\boldsymbol{r}, t)) \tag{8}$$

For the chosen Ω, these two conditions uniquely determine f^0. Since f^0 is an equilibrium, it is in the null space of Ω, and so we can write $f^0 = Ae_0+Be_1+Ce_2$. Then (7) and (8) together provide 3 independent equations in A, B, and C. The result is:

$$f_i^0(\boldsymbol{r},t) = \begin{cases} h(\boldsymbol{r},t)(1-2K) & i=0 \\ (h(\boldsymbol{r},t)/2)\left[K + (\boldsymbol{v_i}\cdot\boldsymbol{u}(\boldsymbol{r},t))/v^2\right] & i=1,2,3,4 \end{cases} \tag{9}$$

The continuity equation is now at hand. We insert (5) and (6) into (4), then sum (4) over $i = 0, 1, ..., 4$, divide by τ, and equate coefficients of ϵ^1. We obtain

$$\left(\frac{\partial}{\partial q_x}, \frac{\partial}{\partial q_y}\right) \cdot \sum_{i=0}^{4} \boldsymbol{v_i} f_i^0(\boldsymbol{r},t) + \frac{\partial}{\partial s}\sum_{i=0}^{4} f_i^0(\boldsymbol{r},t) = 0$$

and so, after multiplying by ϵ,

$$\partial h(\boldsymbol{r},t)/\partial t + \nabla_{\boldsymbol{r}} \cdot [h(\boldsymbol{r},t)\boldsymbol{u}(\boldsymbol{r},t)] = 0 \tag{10}$$

4.2 The Euler Equation

If we multiply (4) by $\boldsymbol{v_i} = (v_{ix}, v_{iy})$, sum over $i = 0, 1, ..., 4$, divide by τ, and again equate coefficients of ϵ^1, we obtain a pair of equations:

$$\frac{\partial}{\partial s}\sum_{i=0}^{4} v_{i\alpha} f_i^0(\boldsymbol{r},t) + \frac{\partial}{\partial q_x}\sum_{i=0}^{4} v_{i\alpha} v_{ix} f_i^0(\boldsymbol{r},t) + \frac{\partial}{\partial q_y}\sum_{i=0}^{4} v_{i\alpha} v_{iy} f_i^0(\boldsymbol{r},t) = 0 \quad \alpha \in \{\text{x,y}\} \tag{11}$$

where the right hand side vanishes due to conservation of momentum. This pair can be expressed as

$$\frac{\partial}{\partial t}[h(\boldsymbol{r},t)\boldsymbol{u}(\boldsymbol{r},t))] + \nabla_{\boldsymbol{r}} \cdot \Pi^0(\boldsymbol{r},t)) = \mathbf{0} \tag{12}$$

where Π^0 denotes the momentum tensor based on the local equilibrium, f^0. This is the Euler equation. We have already observed that the momentum tensor is diagonal, and now the explicit expression for f^0 in (9) allows an important simplification. We have $\Pi^0_{xx} = \Pi^0_{yy} = Kv^2 h(\boldsymbol{r},t)$, and so

$$\frac{\partial}{\partial t}[h(\boldsymbol{r},t)\boldsymbol{u}(\boldsymbol{r},t))] + Kv^2\nabla_{\boldsymbol{r}} h(\boldsymbol{r},t) = \mathbf{0} \tag{13}$$

4.3 The Wave Equation

If we differentiate (10) with respect to t, differentiate (13) with respect to $\boldsymbol{r}$, and subtract, we obtain

$$\partial^2 h(\boldsymbol{r},t)/\partial t^2 - Kv^2\nabla^2_{\boldsymbol{r}} h(\boldsymbol{r},t) = \mathbf{0} \tag{14}$$

the classical wave equation with wave speed $v\sqrt{K}$.

To this point, our derivation is similar in spirit to that of Chopard and Droz [2], but we have avoided the complexity of their approach by using an explicit collision matrix and a single time scale, rather than the more conventional, relaxation equation with multiple time scales. Note that although (14) gives us only a constant-speed wave, that speed is controllable by selection of the collision matrix parameter, K.

5 Dispersion and Wave Number Spectra

In the standard model of deep-water waves, wave speed (phase velocity) is given by $\sqrt{g/k}$, where g is the gravitational constant and k is the *wave number*, the spatial analogue of frequency with units m^{-1} [20]. Note that phase velocity, $\sqrt{g/k}$, yields wave frequency $\sqrt{gk}$. If $\Omega(K)$ denotes the collision matrix of (3), then given a target wave number, k, we can use $\Omega(g/(v^2k))$ in the update equation (2) to achieve the desired wave speed. An accurate model of a large body of water is likely to have multiple wave numbers per site. Such composite waves disperse with time according to their component wave numbers. Since (2) describes only local, per site collisions, our strategy is to adjust Ω per wave number to control speeds.

If the wave numbers present in a given wave height field, $h(\boldsymbol{r})$, are not evident from the height field construction, it is straightforward to estimate them. If the underlying process is wide-sense stationary, then the lag $\boldsymbol{r}$ auto-covariance function of the wave height field is given by

$$R(\boldsymbol{r}) = E[h(\boldsymbol{x})h(\boldsymbol{x}+\boldsymbol{r})]$$

where E is the expected value operator. The *wave number spectrum* is then the Fourier transform

$$\phi(\boldsymbol{k}) = \frac{1}{(2\pi)^2}\int R(\boldsymbol{r})e^{-i\boldsymbol{k}\cdot\boldsymbol{r}}d\boldsymbol{r}$$

which carries the amount of energy in, and hence importance of, the waves at each wave vector. The wave number is the modulus of the wave vector. If the wave height field is specified on a lattice, we can use the sample auto-covariance sequence and estimate the wave number spectrum as its discrete Fourier transform (DFT).

In the absence of obstructions, water waves can maintain their speeds (and hence wave numbers) for great distances, sometimes hundreds of miles [20]. To update a composite wave at any given site, we need to apply multiple update matrices, $\Omega(K)$, one to each wave component. We thus maintain total site density, $h(\boldsymbol{r})$, in terms of its wave-number-indexed components,

$$h(\boldsymbol{r},t) = \sum_k \sum_{i=0}^{4} f_{i,k}(\boldsymbol{r},t) \tag{15}$$

and we apply $\Omega(g/(v^2k))$ to update the $f_{i,k}(\boldsymbol{r},t)$ as in (2). In the absence of obstructions that change wave numbers (described in Section 7), we can treat the wave-number-indexed components independently.

This opens the issue of how many wave numbers will be needed for visually accurate representation of interesting surfaces. If the height field is centered on a lattice of edge dimension N, then by symmetry alone we need at most $N(N+2)/8$ wave numbers, one for each lattice point in a 45-degree octant. Of course, some circles about the origin will contain more than one such lattice point. The number of distinct radii among all circles through all lattice points is asymptotically $0.764 \times (N/2-1)^2/\sqrt{2log(N/2-1)}$ [21] (cited in [22]). To represent all of them would require both excessive storage and computation time.

Instead, we observe that if we restrict our reflection model (Section 7) to first and second order effects, wave numbers will either remain constant or double on each update. Thus $logN$ wave numbers (powers of 2 in lattice units) should suffice, which yields a total update effort of $O(N^2 logN)$, identical to that of the fast Fourier transform.

6 Initial Conditions

Initial conditions can be arbitrary, but if the goal is to model naturally occurring water waves, we are obliged to begin with a height field that is a reasonable representation of such. Thus, we initially ignore any wave obstructions and begin with a known solution to the general wave equation, in particular, a finite, weighted sum of cosine functions,

$$h(\boldsymbol{r},0) = \frac{1}{N^2}\sum_{\boldsymbol{k}} w(\boldsymbol{k}) e^{2\pi i \boldsymbol{k}\cdot\boldsymbol{r}} \tag{16}$$

where the weights, $w(\boldsymbol{k})$, are specified in frequency space, and the height field is given by the DFT. For the field to be real, we must have the conjugate $w^*(\boldsymbol{k}) = w(-\boldsymbol{k})$, where positions are interpreted mod N. We follow Tessendorf [17] and enforce this constraint by taking

$$w(\boldsymbol{k}) = w_0(\boldsymbol{k}) + w_0^*(-\boldsymbol{k}) \tag{17}$$

where $k = |\boldsymbol{k}|$ and $w_0(\boldsymbol{k})$ is calculated from the targeted wave number spectrum, $\phi(\boldsymbol{k})$. The Phillips spectrum [23] is a standard choice. We use a slightly modified version and instead take

$$w_0(\boldsymbol{k}) = (C\sqrt{e^{-1/k^2}}/k^2)(N(0,1) + iN(0,1))((\boldsymbol{k}/k)\cdot\boldsymbol{D}) \tag{18}$$

where $N(0,1)$ denotes a random sample from a standard normal distribution, D is the wind direction, and C is a scaling constant.

We can write (16) in terms of individual wave numbers as

$$h(\boldsymbol{r},0) = \sum_{k}\sum_{|\boldsymbol{k}|=k}\frac{1}{N^2} w(\boldsymbol{k}) e^{2\pi i \boldsymbol{k}\cdot\boldsymbol{r}} \tag{19}$$

and again treat each wave number independently. Comparing (15), we see that for each site, $\boldsymbol{r}$, and each wave number, k, we need to specify values $f_{i,k}(\boldsymbol{r},0)$ so that

$$\sum_{|\boldsymbol{k}|=k} \frac{1}{N^2} w(\boldsymbol{k}) e^{2\pi i \boldsymbol{k}\cdot\boldsymbol{r}} = \sum_{i=0}^{4} f_{i,k}(\boldsymbol{r},0) \tag{20}$$

The specification of these values is otherwise open, but we find the most compelling wave action to arise if we first decompose the wind direction, $\boldsymbol{D}$, into its associated positive lattice directions, f_{i_1} and f_{i_2}. We then select that vector, $\boldsymbol{k}$, having maximum dot product, $\boldsymbol{D}\cdot\boldsymbol{k}$, and distribute the entire left hand side of (20) to f_{i_1} and f_{i_2} in proportion to the components of $\boldsymbol{k}$.

7 Obstructions

In addition to their computational simplicity, a widely recognized advantage of LB methods over conventional (finite element, finite difference) methods is their ability to handle complicated boundary conditions. We can represent the collision of a wave with an obstruction by simply reflecting the directional density, dissipating its amplitude, and if harmonics are desired, doubling its wave number. For example, if $\boldsymbol{r}$ is a site adjacent to an obstruction at $\boldsymbol{r}+(\lambda,0)$, then the update at $\boldsymbol{r}$ at time t, which would have routed density to $f_{1,k}(\boldsymbol{r}+(1,0),t+\tau)$ will instead route a possibly reduced amount to $f_{3,k+s}(\boldsymbol{r}-(1,0),t+\tau)$, where s is either 0 or k. The flow reduction, if any, represents energy dissipation.

8 Implementation

We implemented both the lattice-Boltzmann wave model and a ray-tracing renderer in OpenCL. They can be executed simultaneously on a single NVIDIA GTX 480. Individual frames from a sample animation are shown in Figure 2. For this animation, we used a 1024×1024 grid with 16 wave numbers, and we were able to render 1024×768-pixel frames at 25 frames per second.

The OpenCL kernel for the wave model is nearly trivial, as should be expected from the update equation (2). The only item of note is that the directional density storage, which requires WIDTH×DEPTH×DIRECTIONS floats, is implemented as

```
#define store(i,j,k) ((i)*(WIDTH*DIRECTIONS)+(k)*WIDTH+(j))
```

so that the WIDTH index, rather than the DIRECTIONS index, varies most rapidly in linear memory. There are 5 directions, but the width is usually a large power of 2, and so this storage alignment allows the NVIDIA architecture to fully coalesce accesses to device (card) memory, which is important to performance.

The ray-tracing renderer is based largely on the approach developed by Musgrave [24], as this algorithm lends itself well to GPU computation. Unlike kd-tree traversals of large sets of triangles, there is no control-flow based on the direction of the cast ray, which allows all rays to follow the same execution path

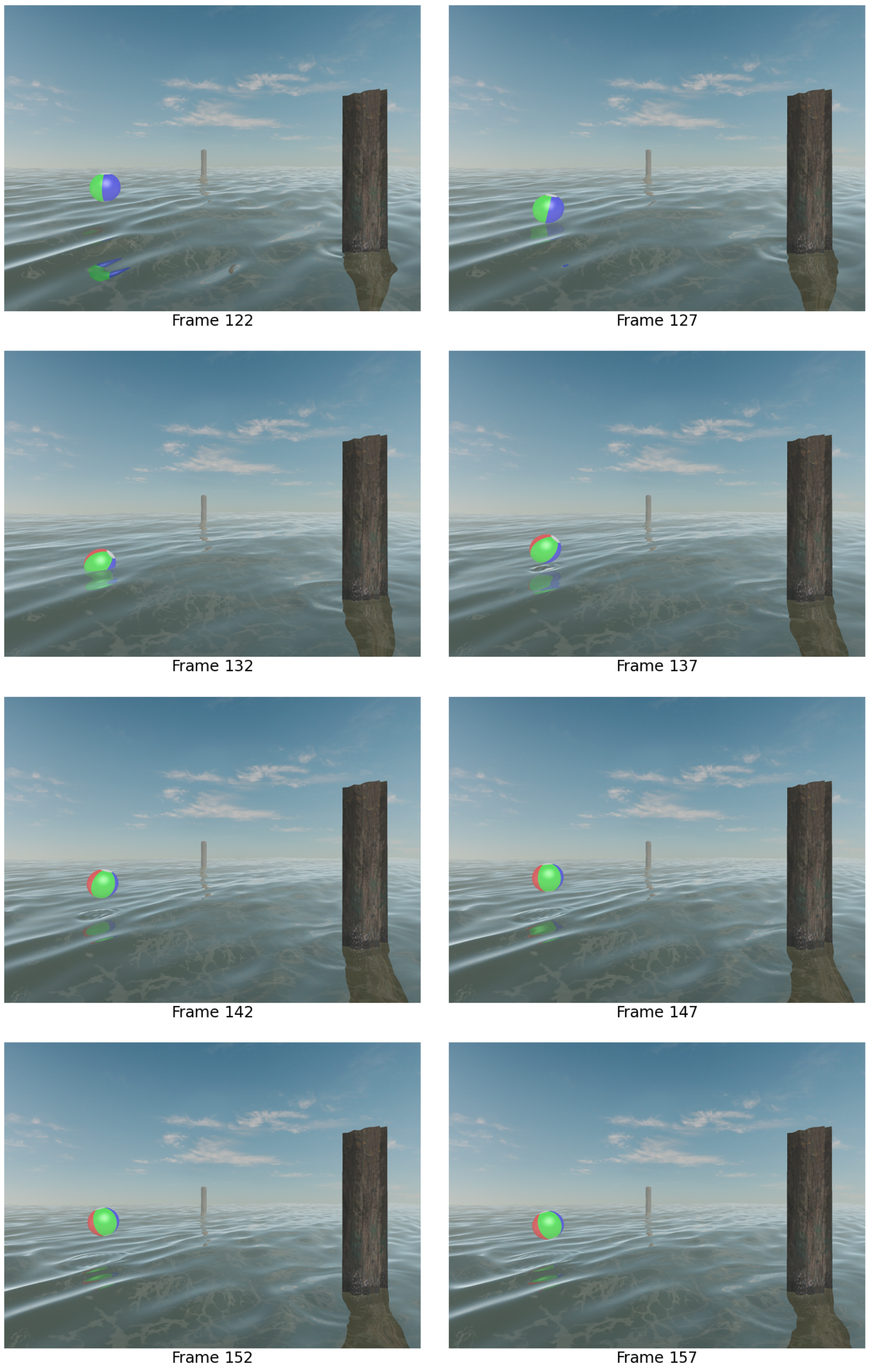

Fig. 2. Frames from sample animation.

until a potential hit is encountered. The increased coherence allows the GPU to compute a larger number of rays in parallel, thereby enabling real-time frame rates.

The entire lattice-Boltzmann grid represents a height map. This height map is stored in the red component of a texture object, since NVIDIA's architecture caches accesses to textures. A modified Bresenham Digital Differential Analyzer (DDA) algorithm [25] is then used for the traversal. Once a potential intersection point is found, triangles representing the cell that is intersected are generated from the height map, taking advantage of the spatial locality of the texture cache. A standard ray-triangle intersection test is used. Once all rays have been tested for intersection against the water surface, intersection tests for the channel markers and the beach ball are carried out using traditional ray-tracing methods for reflection, refraction, transmission, and occlusion with respect to the water.

9 Conclusions

We have suggested a new technique for modeling deep-water waves that is based on a two-dimensional, lattice-Boltzmann method. It includes wave dispersion and offers a flexible facility for handling wave-object interaction. Modeling and rendering can be carried out simultaneously, in real-time, on a single graphics card.

Extensions currently under investigation include wave interaction with boats or other partially submerged, moving objects and wave interaction with porous materials.

Acknowledgments

This work was supported in part by the U.S. National Science Foundation under Award XXXXXXX and by an equipment donation from NVIDIA Corporation. Thanks to the Peabody Symphony Orchestra, Hajime Teri Murai, conductor, for use of an excerpt from the second movement of Debussy's *La Mer* in the accompanying video.

References

1. Gardner, M.: Mathematical games: John conway's game of life. Scientific American (1970)
2. Chopard, B., Droz, M.: Cellular Automata Modeling of Physical Systems. Cambridge Univ. Press, Cambridge, UK (1998)
3. Geist, R., Steele, J., Westall, J.: Convective clouds. In: Natural Phenomena 2007 (Proc. of the Eurographics Workshop on Natural Phenomena), Prague, Czech Republic (2007) 23 – 30, 83, and back cover
4. Geist, R., Westall, J. In: Lattice-Boltzmann Lighting Models. Volume 1. (2010)

5. Thürey, N., Rüde, U., Stamminger, M.: Animation of open water phenomena with coupled shallow water and free surface simulations. In: SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation, Vienna, Austria (2006) 157–164
6. Wei, X., Li, W., Mueller, K., Kaufman, A.: The lattice-boltzmann method for gaseous phenomena. IEEE Transactions on Visualization and Computer Graphics **10** (2004)
7. Kass, M., Miller, G.: Rapid, stable fluid dynamics for computer graphics. In: SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM (1990) 49–57
8. Foster, N., Metaxas, D.: Realistic animation of liquids. Graph. Models Image Process. **58** (1996) 471–483
9. Foster, N., Fedkiw, R.: Practical animation of liquids. In: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM (2001) 23–30
10. Stam, J.: Stable fluids. In: SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press/Addison-Wesley Publishing Co. (1999) 121–128
11. Mastin, G., Watterberg, P., Mareda, J.: Fourier synthesis of ocean scenes. IEEE Computer Graphics and Applications **7** (1987) 16–23
12. Hasselmann, D.E., Dunckel, M., Ewing, J.A.: Directional wave spectra observed during jonswap 1973. Journal of Physical Oceanography **10** (1980) 1264–1280
13. Jensen, L.: Deep-water animation and rendering. http://www.gamasutra.com/gdce/2001/jensen/jensen_pfv.htm (2001)
14. GmbH, C.: Cryengine3 specifications. http://www.crytek.com/technology/cryengine-3/specifications/ (2010)
15. Hinsinger, D., Neyret, F., Cani, M.P.: Interactive animation of ocean waves. In: SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, New York, NY, USA, ACM (2002) 161–166
16. Yuksel, C., House, D.H., Keyser, J.: Wave particles. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007) **26** (2007) 99
17. Tessendorf, J.: Simulating ocean water. In: Simulating Nature: Realistic and Interactive Techniques, SIGGRAPH '01 Course #47 Notes, Los Angeles, CA (2001)
18. Tessendorf, J.: Interactive Water Surfaces. In: Game Programming Gems 4. Charles River Media, Rockland, MA, USA (2004)
19. Salmon, R.: The lattice boltzmann method as a basis for ocean circulation modeling. Journal of Marine Research **57** (1999) 503535
20. Kinsman, B.: Wind Waves: their generation and propagation on the ocean surface. Prentice-Hall, Englewood Cliffs, NJ (1965)
21. Landau, E.: Über die einteilung der positiven ganzen zahlen in vier klassen nach der mindestzahl der zur ihrer additiven zusammensetzung erforderlichen quadrate. Archiv der Math. und Physik **13** (1908) 305–312
22. Moree, P., te Riele, H.J.J.: The hexagonal versus the square lattice. Math. Comput. **73** (2004) 451–473
23. Phillips, O.: On the generation of waves by turbulent wind. Journal of Fluid Mechanics **2** (1957) 417445
24. Musgrave, F.: Grid tracing: Fast ray tracing for height fields. Technical Report RR-639, Yale University, Dept. of Comp. Sci. (1988)
25. Bresenham, J.E.: Algorithm for computer control of a digital plotter. IBM Systems Journal **4** (1965) 2530