



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

## *Featured-Based Vector Simulation of Water Waves*

Qizhi Yu — Nathalie Praizelin — Frank Rochet — Fabrice Neyret

N° 6855

Janvier 2009

Thème COG

 *Rapport  
de recherche*



## Featured-Based Vector Simulation of Water Waves

Qizhi Yu <sup>\*</sup>, Nathalie Praizelin <sup>†</sup>, Frank Rochet <sup>†</sup>, Fabrice Neyret <sup>‡\*</sup>

Thème COG — Systèmes cognitifs  
Équipes-Projets Evasion

Rapport de recherche n° 6855 — Janvier 2009 — 22 pages

**Abstract:** We present a feature-based vector simulation approach for simulating local wave phenomena which often appear in running streams. By using this approach, we simulate the quasi-stationary waves caused by obstacles in shallow water. The geometric features of the waves are constructed and animated using vector representation. The vector data is then converted to feature aligned meshes for capturing the shape of the waves. Our approach provides high-resolution details with real-time performance, and allows users to intuitively control the simulation behavior.

**Key-words:** Rivers, wave simulation, animation, phenomenological simulation, vector representation, real-time, natural phenomena.

\* LJK/IMAG-INRIA

† GRAVIR/IMAG-INRIA

‡ CNRS

## **Simulation vectorielle des structures d'ondes de surface**

**Résumé :** Nous proposons une approche basée sur la simulation des structures apparentes sous forme vectorielle pour simuler les divers phénomènes d'ondes qui apparaissent à la surface des ruisseaux et rivières. En particulier, nous simulons à l'aide de cette approche le motif de vagues à l'amont des obstacles dans le flot. Pour cela, nous proposons une construction géométrique efficace basée sur les lois de l'hydrodynamique, qui permet de générer et d'animer sous forme vectorielle les structures apparentes ciblées. A partir de ces informations vectorielles, nous générons un maillage local aligné sur chaque structure pour capturer efficacement à haute résolution les détails des ondes locales. Nos résultats montrent que la méthode est bien adaptée aux applications temps-réel. De plus, l'approche permet un contrôle intuitif de paramètres par un utilisateur.

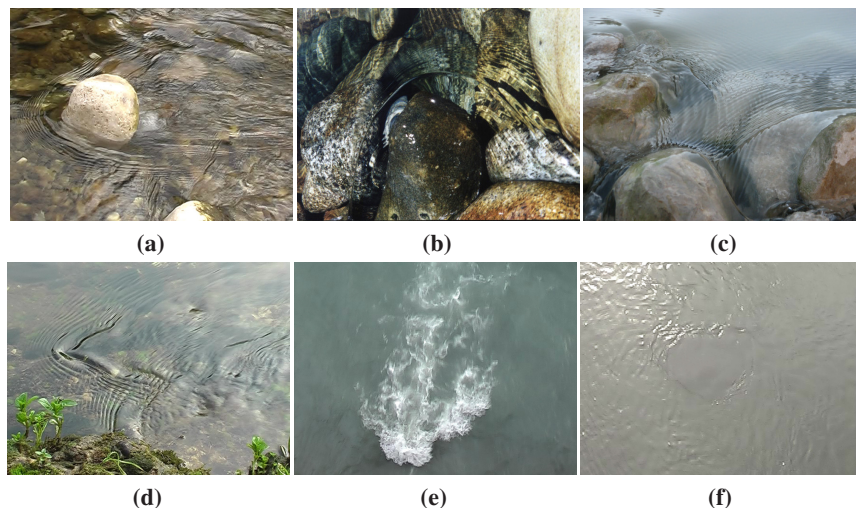
**Mots-clés :** Rivières, simulation de vagues, animation, simulation phénoménologique, représentation vectorielle, temps réel, phénomènes naturels.

## 1 Introduction

Simulating fluids (*e.g.*, liquids, gases) is an important topic in computer graphics because it has a wide range of applications, including video games, film effects and simulators. For water, many efforts have been put on simulating surface waves. This is because the most distinct visual impression of large bodies of water in the natural world (*e.g.*, river, lake, and sea) is the distortion of reflection and refraction due to the deformation of the water-air surface.

To date, studies on the simulation of water waves have yielded many realistic results, but many local wave phenomena (Fig. 1) that can be often found in running streams still can not be easily handled by existing methods. Simulating these phenomena are challenges especially when we target real-time interactive applications where users are allowed to dynamically decide to observe any feature in a very close view or in a very wide view. Relying solely on Computational Fluid Dynamics (CFD) simulations [FF01] would require very high resolution discretization, which is not compatible with the large scale of the background fluid domain and the real-time context. Moreover, poor controllability is another shortcoming of using CFD techniques. There is no intuitive relation between the control variables (*e.g.*, initial and boundary conditions) and the simulation behavior. On the other hand, there exists some non-numerical simulation methods for simulating water waves that are often fast and stable [Tes04, YHK07]. But it is not clear how to extend these methods to simulate the waves we focus on.

In this paper, we propose a feature-based vector simulation approach in order to simulate the local waves with given flow data (2D horizontal velocity and depth) of streams. Note that the resolution of the input flow data can be much coarser than the wave details we want to present. Our approach is based on the observation that most of the local wave phenomena have quasi-regular and quasi-stationary geometric structures. The idea is to separate the representation for animation and rendering. We use



**Figure 1:** Local wave phenomena in water flow: (a,b) waves caused by obstacles in shallow water, (c) intersection of waves caused by obstacles, (c) waves caused by underwater obstacles in deep water, (d) hydraulic jump and foams, and (e) boils (note the circular feature in the middle).

vector representation to capture the geometric feature and motion behavior of the target waves. The concise vector representation allows low cost animation and provides good controllability. The simulated vector information are then converted to high resolution representation, such as meshes, which is suitable for high-quality rendering. Since the high-resolution is only used for local features, we still can ensure high performance.

We apply the presented approach to simulate the wave pattern caused by the obstacles in shallow streams (Figs. 1a and 1b). It is a common phenomenon in nature, but no existing method in computer graphics could ever simulate it. In this case, we treat wave crests as the geometric feature of the wave pattern. We construct the wave crests by using the knowledge from classic hydrodynamics without numerical simulation. This geometric construction technique is mostly based on our previous work [NP01], but we have improved it for better robustness and performance. For high-quality rendering, we build feature aligned meshes along the wave crests with user defined wave profiles. Moreover, we handle wave intersection (Fig. 1c) that often appears between two close obstacles. Our results demonstrate that the presented approach provides high resolution details and real-time performance.

The rest of the paper is organized as follows. After briefly summarizing related work, we describe the input data of our model in Section 3. The two parts of our method: vector simulation and surface construction are presented in Sections 4 and 5, respectively. Section 6 gives implementation details. We demonstrate our results in Section 7. Finally, we conclude with possible future work in Section 8.

## 2 Related work

State-of-the-art water animation techniques generally rely on numerically solving full 3D Navier-Stokes equations [FM96, Sta99] and tracking the water surfaces with the level set method [FF01, EMF02]. Though the performance of these techniques can be improved by using adaptive grids [LGF04, IGLF06], they are still not practical in real-time graphics, especially for handling the surface waves of large bodies of water.

One way of reducing the computational complexity is to neglect the 3D features of waves, *i.e.*, to model the water surfaces with height fields. In this case, various simplified fluid equations could be derived from 3D Navier-Stokes equations with different assumptions. Kass and Miller [KM90] used linear wave equation to simulate waves with small amplitude in shallow water. Layton and van de Panne [LvdP02] simulated waves by solving 2D shallow water equations with a semi-Lagrangian time integration method. Chen *et al.* [CdVLHM97] used the pressure solved from 2D Navier-Stokes equations to modulate the height of water surfaces. In a small bounded domain, these methods are applicable for generating waves with limited resolution in real-time interactive applications.

Instead of numerically simulating fluid motion, some wave models reconstruct the water surfaces directly from the geometric or statistical properties of waves. This kind of approach appeared earlier than did the physically-based simulation in graphics communities, but it is still the first choice in many today's industry applications due to their advantages of fast and stable computation, low memory cost and high controllability. Perlin noise [Per85] can be used for modeling random waves with very low computational cost. Classic procedural wave models approximate deep water waves by superposing sinusoidal functions [Max81, FR86, Pea86]. By introducing an adaptive scheme, Hinsinger *et al.* [HNC02] achieved interactive animation of unbounded ocean surfaces. Instead, FFT wave models [MWM87, Tes04] perform the superposition in

the spectrum domain, and thus the statistical wave spectrum of real ocean can be easily applied.

There existing some wave models which focus on local wave phenomena. Glassner [Gla02] simulated ship waves by geometric construction. However, in his method the interaction between the waves and other floating perturbations (*e.g.*, leaves) is not possible. Most recently, Yuksel *et al.* [YHK07] proposed a concept called wave particles to efficiently simulate surface waves triggered by object-water interaction. However, this method does not treat the waves related to water current.

### 3 Input data

The physical cause of the wave phenomenon we focus on is related to the local flow conditions such as flow velocity and water depth. We take this data as the input of our simulation.

Since everyday rivers are calm, we can model the river flow with the superposition of a steady mean flow and time-varying perturbations. We precompute the steady velocity  $\mathbf{v}_s$  and water depth  $h$  by solving the shallow water equations with the finite volume method [Wu04]. Note that the resolution of simulation could be much coarser than the resolution required by the rendering. To enhance the dynamic details of the velocity field, we make the steady velocity field quasi-stationary by superimposing the velocity fields of local perturbations  $\mathbf{v}_{p_i}$  on it as in [WH91]:

$$\mathbf{v} = \mathbf{v}_s + \sum_i \mathbf{v}_{p_i}. \quad (1)$$

For example,  $\mathbf{v}_{p_i}$  can be the analytic solution of sink, source or vortex with a cutoff radius. Note that these perturbations should be advected with the stationary flow  $\mathbf{v}_s$ .

### 4 Vector simulation

We assume that the predominant stationary wave caused by an obstacle in a shallow stream can be approximated by the shallow water theory [Sto57, page 22], which suggests that its wave speed is independent of its wavelength:

$$|\mathbf{c}| = \sqrt{gh}, \quad (2)$$

where  $g$  is the gravitational acceleration and  $h$  is the local water depth. Using the analogy of the shallow water theory with compressible gas dynamics, we call the wave *shockwave* in the following discussion. In addition, we assume that the shockwave triggers a series of ripples upstream of it. Fig. 2 gives a schematic illustration of our assumptions. In the following, we examine the geometric properties of the shockwave.

In a running stream, a wave can be stationary provided the wave speed

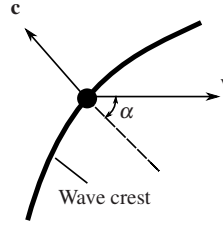
$$|\mathbf{c}| = |\mathbf{v}| \cos \alpha, \quad (3)$$

where  $\mathbf{v}$  is the local flow velocity and  $\alpha$  is the propagation angle of the wave relative to the upstream direction (Fig. 3). Therefore, the crest of a shockwave should lie at an angle

$$\alpha = \arccos \frac{\sqrt{gh}}{|\mathbf{v}|} \quad (4)$$



**Figure 2:** We assume that the waves made by an obstacle consist of a predominant shockwave and a series of ripples.



**Figure 3:** A wave that propagates at the angle of  $\alpha$  to the upstream direction. Here,  $c$  is the wave velocity, and  $v$  is the local flow velocity.

to the upstream direction.

Now, let us examine where the most upstream point, *starting point*, of a shockwave locate (Fig. 4). At the starting point, the wave crest should be orthogonal to  $v$ , *i.e.*,  $\alpha = 0$ . Substituting it into Eq. 4, we get  $|v| = \sqrt{gh}$ . To facilitate further discussion, we remind the Froude number:

$$F_r = \frac{|v|}{\sqrt{gh}}. \quad (5)$$

A supercritical flow ( $F_r > 1$ ) past an obstacle will leave a subcritical area ( $F_r < 1$ ) in front of the obstacle (Fig. 4). This means we can expect to find a starting point at the boundary of the subcritical area where  $F_r = 1$ <sup>1</sup>.

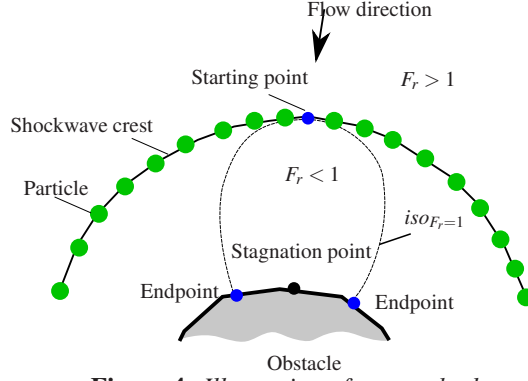
Based on the above observation, we can construct the shockwave crest in front of an obstacle as follows (Fig. 4):

1. Identify a subcritical region upstream from the obstacle by constructing a contour line of  $F_r = 1$  (Section 4.1).
2. Find the starting point of the shockwave along this contour line (Section 4.2).
3. Generate particles at the starting point and advect them to trace the shockwave crest according to Eq. 4 (Section 4.3).

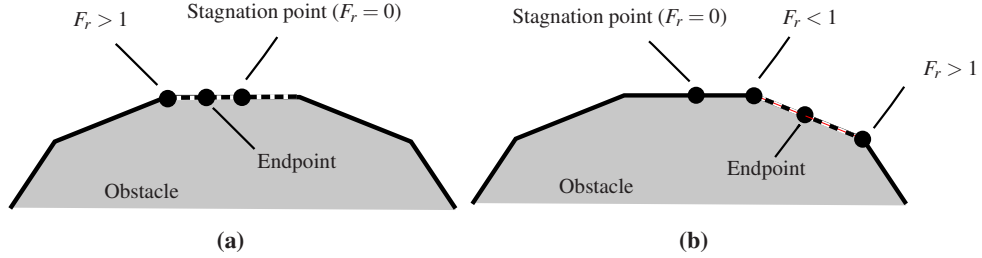
For animating the wave, we update the starting point at every frame according to local time-varying velocity (Section 4.4).

<sup>1</sup>As stated in [NP01], it is also possible to find a starting point downstream of an obstacle. For clarity, we only consider the upstream case in this paper, but the downstream case is mostly identical.





**Figure 4:** Illustration of our method.



**Figure 5:** Along the boundary of an obstacle, finding edges which contain the endpoints of  $iso_{F_r=1}$ . It might be (a) an edge which contains the stagnation point and  $F_r > 1$  at one of the two ends, or (b) an edge which does not contain the stagnation point but  $F_r > 1$  at one end and  $F_r < 1$  at the other end.

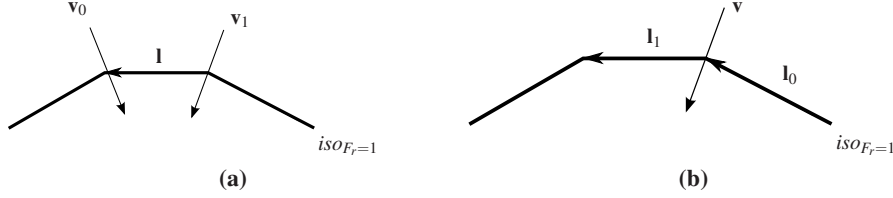
#### 4.1 Constructing the contour line of $F_r = 1$

When a flow passes an obstacle, there must be a point, called *stagnation point*, along the boundary of the obstacle where  $|\mathbf{v}| = 0$ , i.e.,  $F_r = 0$ . In addition, there are two *endpoints*, where  $F_r = 1$ , on the two sides of the stagnation point (Fig. 4). Starting from one of the endpoints, we can trace a contour line in the triangular mesh of the water surface from cell to cell by linking linearly interpolated points on cells' edges where  $F_r = 1$ . The tracing procedure stops until it reaches another endpoint. Finally, we can get a contour line that consists of straight line segments by connecting these points.

To find the endpoint, we search edges along the boundary of the obstacle, starting from the edge containing the stagnation point. For the edge containing the stagnation point, if  $F_r \geq 1$  at one of the nodes of the edge, we can find an endpoint on this edge between the stagnation point and the node by interpolating (Fig. 5a). For other edges, an endpoint exists if and only if  $F_r \leq 1$  at one node and  $F_r \geq 1$  at the other node (Fig. 5b).

#### 4.2 Finding the shockwave starting point on a contour line

As stated earlier, the wave starting point should lie on the contour line of  $F_r = 1$ , and the flow velocity  $\mathbf{v}$  should be orthogonal to the wave crest at the starting point. Furthermore,



**Figure 6:** The starting point of a shockwave can be found on the contour line of  $F_r = 1$ . Since the contour line consists of line segments, the starting point may locate (a) on a segment where  $(\mathbf{v}_0 \cdot \mathbf{l}) \cdot (\mathbf{v}_1 \cdot \mathbf{l}) \leq 0$ , or (b) at a node where  $(\mathbf{v} \cdot \mathbf{l}_0) \cdot (\mathbf{v} \cdot \mathbf{l}_1) \leq 0$ .

the wave crest should be tangent to the contour line at the starting point, otherwise the wave crest will enter into the subcritical area where  $F_r < 1$  and Eq. 2 can not be satisfied. Therefore, we can find the starting point along the contour line of  $F_r = 1$  by searching the location where flow velocity  $\mathbf{v}$  is orthogonal to the contour line.

Since the contour line consists of line segments in practice, we find the starting point by approximation. We first try to find a segment with direction  $\mathbf{l}$  which satisfies:

$$(\mathbf{v}_0 \cdot \mathbf{l}) \cdot (\mathbf{v}_1 \cdot \mathbf{l}) \leq 0, \quad (6)$$

where  $\mathbf{v}_0$  and  $\mathbf{v}_1$  are the flow velocities at the two ends of the segment (Fig. 14a). Once having found such a segment, we interpolate along this segment to find the starting point. If no such segment is found, we use the node which satisfies

$$(\mathbf{v} \cdot \mathbf{l}_0) \cdot (\mathbf{v} \cdot \mathbf{l}_1) \leq 0 \quad (7)$$

as the starting point. Here,  $\mathbf{v}$  is the flow velocity at the node in question, and  $\mathbf{l}_0$  and  $\mathbf{l}_1$  are the directions of the two segments connected to this node, respectively (Fig. 14b).

### 4.3 Tracing wave crest with particles

Once having obtained the starting point of a shockwave, our task is to trace the wave crest. Before introducing our method, let us first investigate the behavior of the elements of a stationary wave (Fig. 7). In a running stream, waves not only propagate at the wave velocity  $\mathbf{c}$  but also advect with the current of velocity  $\mathbf{v}$ . Hence the resultant velocity of a wave element is the vector sum of the two velocities:

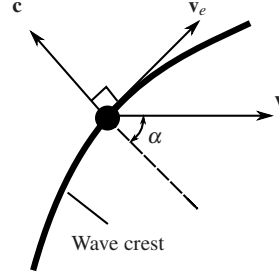
$$\mathbf{v}_e = \mathbf{v} + \mathbf{c}. \quad (8)$$

Eq. 2 tells us that the component  $|\mathbf{v}| \cos \alpha$  of stream velocity at right angles to the crest cancels the crest's motion at the wave speed  $|\mathbf{c}|$ . Thus we have

$$|\mathbf{v}_e| = |\mathbf{v}| \sin \alpha = \sqrt{|\mathbf{v}|^2 - |\mathbf{c}|^2}, \quad (9)$$

and  $\mathbf{v}_e$  is tangent to the wave crest.

Based on the above observation, we propose a method from the Lagrangian point of view. We use particles to represent the wave elements discussed above. At each time step, we generate two new particles respectively at the two sides of a shockwave starting point and update all existing particles with  $\mathbf{v}_e$ . In addition, in order to imitate the attenuation of wave energy, we associate an intensity value to each particle. The intensity value starts from 1 and linearly decrease to 0 in a given time period. We kill a particle once its intensity value vanishes. Finally, connecting existing particles with line segments gives us a shockwave crest (Fig. 4).



**Figure 7:** The velocity of a wave element  $\mathbf{v}_e$  is the vector sum of the wave velocity  $\mathbf{c}$  and the stream velocity  $\mathbf{v}$ . Here, the wave crest propagates at the angle of  $\alpha$  to the upstream direction.

#### 4.4 Animating the waves

In our previous work [NP01], the steps presented in the previous subsections were executed every frame for accounting for the time-varying velocity field. That is to say, we reconstruct the whole contour line of  $F_r = 1$  at each time step. However, this might occasionally yield discontinuous constructions which lead to the popping of wave crests. Since the water flow considered in our problem is quasi-stationary with small perturbations, it is not necessary to reconstruct the contour line of  $F_r = 1$ . For obtaining better robustness and higher efficiency, we construct the contour line and find the corresponding shockwave starting point only once. At every frame, we update the shockwave starting point according to local velocity perturbations.

For a shockwave starting point  $\mathbf{x}$ , we store its initial location  $\mathbf{x}_0$  and a local steady velocity gradient  $d\mathbf{v}_0 = \nabla|\mathbf{v}_s(\mathbf{x}_0)|$ . As shown in Fig. 8, at each time step we update  $\mathbf{x}$  with an offset  $\lambda$  from  $\mathbf{x}_0$  along the direction  $d\mathbf{v}_0$ :

$$\mathbf{x} = \mathbf{x}_0 + \lambda \frac{d\mathbf{v}_0}{|d\mathbf{v}_0|}, \quad (10)$$

to ensure  $|\mathbf{v}(\mathbf{x})| = |\mathbf{c}|$ , *i.e.*,  $Fr = 1$ . Now our goal is to solve  $\lambda$  at each time step. Using linear approximation, we can estimate stationary flow velocity  $\mathbf{v}_s(\mathbf{x})$  with

$$\mathbf{v}_s(\mathbf{x}) \approx \mathbf{v}_s(\mathbf{x}_0) + \lambda d\mathbf{v}_0. \quad (11)$$

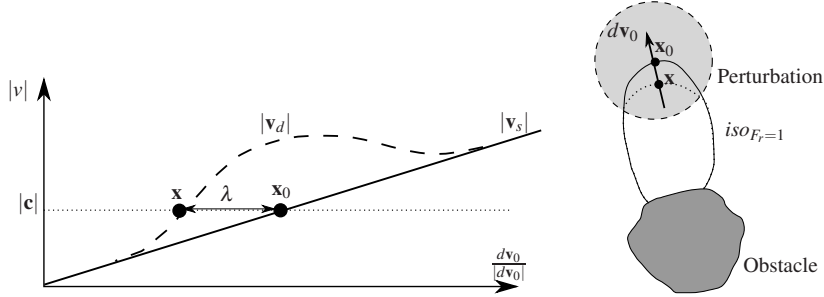
In addition, we assume that only the nearest perturbation  $\mathbf{v}_d$  has a significant influence on the position of the shockwave starting point. Substituting Eqs. 10 and 11 in Eq. 1 yields

$$|\mathbf{v}(\mathbf{x})| = |\mathbf{v}_s(\mathbf{x}_0) + \lambda d\mathbf{v}_0 + \mathbf{v}_p(\mathbf{x}_0 + \lambda \frac{d\mathbf{v}_0}{|d\mathbf{v}_0|})| = |\mathbf{c}| \quad (12)$$

Finally, we can solve the above equation for  $\lambda$  by using a simple iterative scheme.

## 5 Efficient high quality rendering of waves

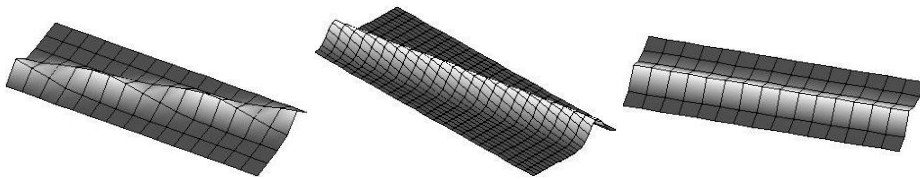
We next turn to the problem of rendering a water surface on which the quasi-stationary waves caused by obstacles are present. We aim at quality rendering with minimum



**Figure 8:** Updating a shockwave starting point  $\mathbf{x}$  from its initial value  $\mathbf{x}_0$  in the vicinity of a perturbation.

memory and computation cost. In real-time graphics, the surface of a large body of water is often represented by a height field. However, this regular grid representation is not optimal in our case. On one hand, the wave phenomenon we focus on is local, *i.e.*, it appears only at some specific locations but not everywhere in the flow domain. Therefore, a wanted representation scheme should be adaptive, *i.e.*, use higher resolution at wave locations and lower resolution elsewhere. On the other hand, as shown in Fig. 9, using a mesh aligned with a feature line is preferable than using a regular grid for reducing geometric aliasing and normal noise [BK01].

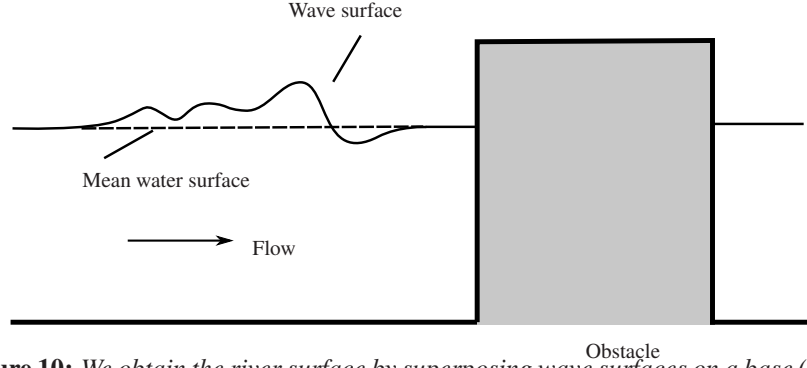
To overcome the shortcomings of height fields, we propose an adaptive representation scheme by exploiting the vector wave information that results from our simulation model. We model the water surface by superposing *wave surfaces* on a *mean water surface* (Fig. 10). As has been stated in Section 3, the mean water surface is treated as the input of our method which is represented by a coarse mesh. Each wave surface is represented by a fine mesh strip aligned to the feature curve of the corresponding wave. In the next subsection, we present how to construct these wave surfaces and superimpose them on the mean water surface. Then, we handle wave intersection in Section 5.2.



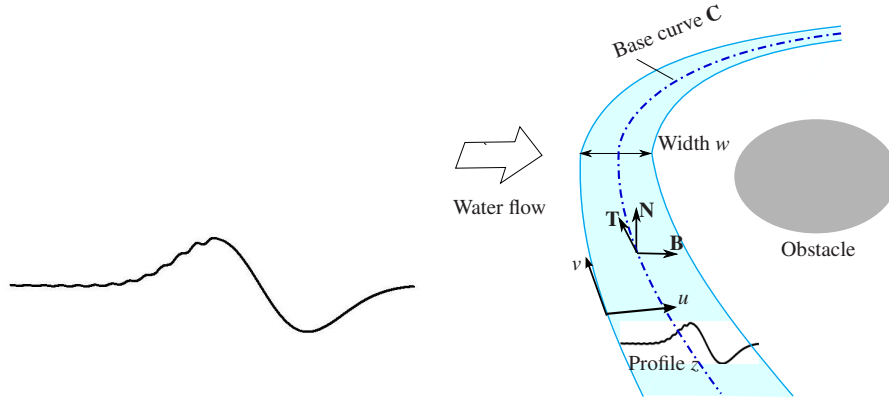
**Figure 9:** Left: Using a regular grid over the domain, cells as thin as a narrow feature can still generate very distracting geometric aliasing. Middle: The sampling rate should be even greater, given by Shannon’s theorem. Right: Using a mesh aligned with the feature line prevents geometric aliasing with fewer cells.

## 5.1 Constructing wave surfaces along shockwave crests

The waves produced by an obstacle in the flow consist of a dominating shockwave and a train of parasitic ripples upstream of it. To construct the wave surface, in theory we need to simulate the crests, profiles and amplitudes of both the shockwave and the ripples. However, what we can obtain now from the simulation model introduced in the



**Figure 10:** We obtain the river surface by superposing wave surfaces on a base (mean) water surface result.



**Figure 11:** Left: Custom wave profile  $z(x)$  defined by Eq. 18. Right: Sweeping the profile curve along a base curve (a shockwave crest) forms a wave surface.

previous section is only the shockwave crests. Fortunately, one important feature of the wave pattern we target is that all the wave crests are nearly regular and parallel. Therefore, it is reasonable to assume that the wave surface to be constructed is the resultant of sweeping a profile curve along a shockwave crest (Fig. 11). Here, the profile curve represents the superposition of one shockwave and a series of ripples. With this assumption, we are able to define the wave surface based on simulated shockwave crests and user given profile curves.

We formulate the surface definition as follows. The sweeping operation uses a shockwave crest as its base curve  $\mathbf{C}(u)$ . Let  $\langle \mathbf{T}, \mathbf{B}, \mathbf{N} \rangle$  be a local frame moving along the base curve, with  $\mathbf{T}$  the unit tangent to the base curve,  $\mathbf{N}$  the normal of the mean water surface, and  $\mathbf{B} = \mathbf{T} \times \mathbf{N}$ . Given a normalized wave profile  $z(v)$ , a wave amplitude function  $a(u)$  and a wave width function  $w(u)$ , we define the parameterized wave surface as:

$$\mathbf{S}(u, v) = \mathbf{S}_b(\mathbf{x}(u, v)) + a(u) \cdot z(v) \cdot \mathbf{N}, \quad (13)$$

where  $\mathbf{S}_b$  is the base water surface and

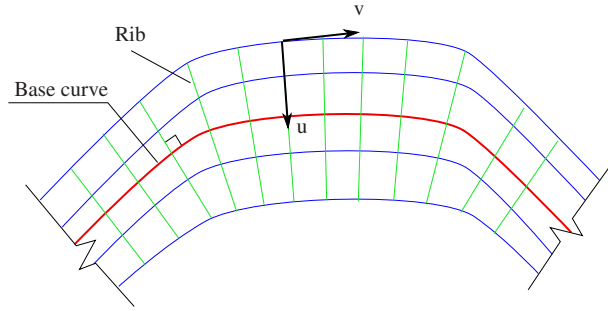
$$\mathbf{x}(u, v) = \mathbf{C}(u) + v \cdot w \cdot \mathbf{B}. \quad (14)$$

If we neglect the derivative of amplitude  $a(u)$ , the surface normal can be calculated by

$$\mathbf{N}_w(u, v) = -\frac{a}{w} \frac{\partial z}{\partial v} \mathbf{B} + \mathbf{N} \quad \text{renormalized.} \quad (15)$$

Note that this formula allows us to compute normals efficiently by using pre-computed derivatives of the wave profile  $z$ .

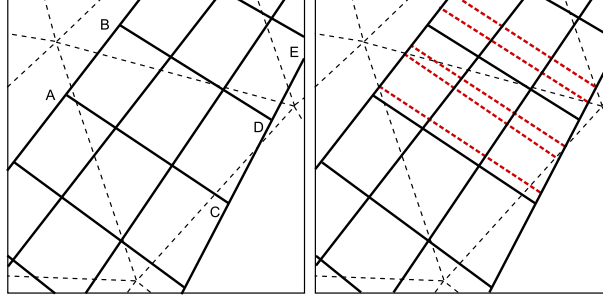
Our goal now is to tessellate the parameterized wave surface,  $\mathbf{S}(u, v)$ , to get a mesh strip that is able to minimize the geometric aliasing. As suggested in [BK01], an effective way to achieve this is to construct a quad mesh whose edges are aligned to iso-parameter lines (Fig. 12). We build the quad mesh in two steps. *First*, we create rib curves uniformly sampled along the base curve. Meanwhile, we let the rib curves be orthogonal to the base curve. Note that the rib curves may intersect each other when the curvature radius of the base curve is less than  $w/2$ . In this case, we relax the requirements of orthogonality. *Second*, we uniformly sample on the rib curves to create lines in another parameterization direction, *i.e.*,  $v$  direction. The sampling density is controlled by the LOD scheme that will be introduced in Section 6.



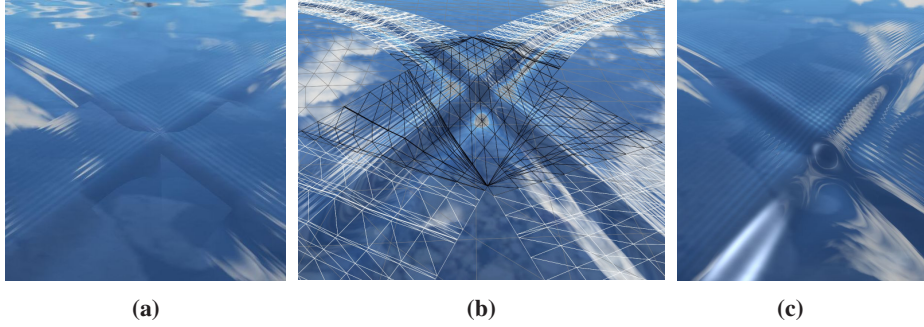
**Figure 12:** Wave surface is sampled by a quad mesh aligned with iso-parameter lines for reducing normal noise.

Once having tessellated the wave surfaces, we need to apply the mesh strips upon the mesh of the mean water surface. Drawing them separately would not give a correct result because wave surfaces may have negative offset from the mean water surface (Fig. 10). The mesh-stitching technique described in [KB00] may work for merging these meshes. However, this method requires remeshing the two surfaces to be merged. Since the waves are dynamic, the remeshing would have to be done in each frame. It is not only computationally expensive but also leads to an extra time cost for uploading the new surface data to the GPU memory.

To avoid remeshing the mean water surface, we solve the mesh composing problem by using the stencil buffer found on graphics hardware. We first draw all wave surfaces into not only a color buffer but also a stencil buffer. The stencil buffer is able to indicate which fragments are covered by the wave surfaces. Then we draw the mean water surface into the color buffer where the wave surfaces are not present. By using this method, we do not need to modify the mean water surface. Still, we need to ensure a perfect continuity between the wave surface and the base water surface. When the boundary of a wave surface intersects with the edges of the mean water surface mesh, geometry gaps may appear. To avoid this, we insert extra ribs by interpolation as shown in Fig. 13.



**Figure 13:** Left: Mean water surface mesh (dash) and original mesh strip (plain). Boundary edges like AB, CD and DE must be split because they intersect with the edges of the mean water surface mesh (dash). Right: The mesh strip with added extra ribs (red dashed lines) by interpolating.



**Figure 14:** The intersection of two wave patterns. (a) A simple Z-buffer rendering without handling crossing. (b) We generate a dedicated mesh patch for the intersection part. (c) Final rendering with a proper intersection treatment.

## 5.2 Handling the intersection of waves

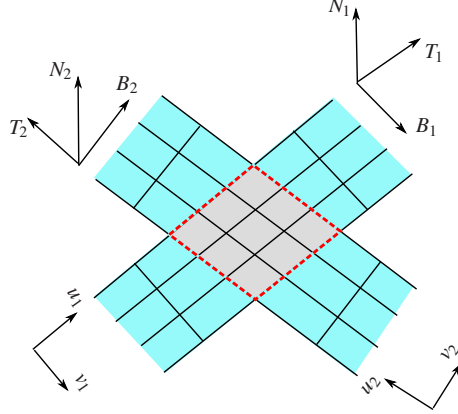
When two obstacles are close, the waves caused by them may intersect and superpose (Fig. 1c). Simply drawing two wave surfaces without handling the intersection will lead to an un-smooth result (Fig. 14a), and can not account for the addition of wave amplitudes. To properly superpose waves, we construct a dedicated mesh patch for the intersection part. Suppose that two wave surfaces  $\mathbf{S}_1(u_1, v_1)$  and  $\mathbf{S}_2(u_2, v_2)$  intersect as shown in Fig. 15. The superposed surface of the intersection part is defined by:

$$\mathbf{S}(u_1, u_2, v_1, v_2) = \mathbf{S}_1(u_1, v_1) + \mathbf{S}_2(u_2, v_2) - \mathbf{S}_b(u_1, u_2). \quad (16)$$

The surface normal can be calculated by:

$$\mathbf{N}_w(u_1, u_2, v_1, v_2) = -\frac{a_1}{w_1} \frac{\partial z}{\partial v}(v_1) \cdot \mathbf{T}_2 + \frac{a_2}{w_2} \frac{\partial z}{\partial v}(v_2) \cdot \mathbf{T}_1 + \mathbf{B}_1 \times \mathbf{B}_2 \quad \text{renormalized.} \quad (17)$$

We tessellate the intersection part with a structured grid aligned with the grid lines of  $\mathbf{S}_1$  and  $\mathbf{S}_2$ . Moreover, we need to cut out this intersection part in both  $\mathbf{S}_1$  and  $\mathbf{S}_2$ .



**Figure 15:** For handling wave intersection, we cut out the intersection part in the two intersected meshes. In addition, we create a dedicated mesh patch (bounded by red dashed lines).

## 6 Implementation details

### 6.1 Wave Profile

The choice of the wave profile  $z(x)$  is up to users except the constraint:  $z(x) = z'(x) = z''(x) = 0$  at the two ends. This constraint ensures  $G^2$  continuity between the wave surface and the base water surface. In our implementation, we used the normalized wave profile illustrated in Fig. 11. It is defined as the sum of a gravity wave profile  $z_g(x)$  and a capillary wave profile  $z_c(x)$  inspired by [FM98]:

$$z(x) = z_g(x) + z_c(x), \quad x \in [-1..1], \quad (18)$$

where

$$z_g(x) = 2\left(3x + \frac{1}{\sqrt{2}}\right)e^{-(3x + \frac{1}{\sqrt{2}})^2}, \quad (19)$$

and

$$z_c(x) = \begin{cases} .045(e^{-2x} \cos(24\pi x) - 1 - x(e^{-2} \cos(24\pi) - 1)), & x \geq 0, \\ 0, & x < 0. \end{cases} \quad (20)$$

### 6.2 Bump-mapping and LOD

Defining a wave surface by sweeping allows us to calculate accurate surface normals efficiently. We always determine per-pixel normals through the analytical bump computed from Eqs. 15 and 17. Problems may occur when we should filter the bumps themselves, *i.e.*, when the sampling frequency (*i.e.* pixel size) is smaller than the frequency of waves. As predicted by Shannon's theorem, this yields aliasing. In practice, this occurs for the capillary wave component of the profile (Eq. 18). So, these high frequency waves must be properly filtered. We progressively fade the capillary ripples according to the view distance  $d$ . We rewrite Eq. 18 to:

$$z(x) = z_g(x) + \beta z_c(x), \quad (21)$$



where  $\beta$  decreases from 1 to 0 as  $d$  increases. The filtered geometric information should be accounted for in the illumination model, which is a general and tough problem. We leave it as future work.

To ensure good performance, we determine the resolution of wave mesh strips according to the view distance  $d$ . We set the level of subdivision of a wave surface in the direction  $u$  as  $\lfloor \log_2((1 - \delta)N_{max}) \rfloor$ , with  $N_{max}$  the maximum number of grid lines, and

$$\delta = \begin{cases} 0, & d \leq d_{near}, \\ \frac{d - d_{near}}{d_{far} - d_{near}}, & d_{near} < d < d_{far}, \\ 1, & d \geq d_{far}. \end{cases} \quad (22)$$

where  $d_{near}$  and  $d_{far}$  correspond to the finest and the coarsest LOD, respectively.

### 6.3 Shading model

For realistic rendering of water surfaces, we rely on existing GPU techniques. We consider reflection (global and local, with Fresnel term) and refraction effects. Local reflection and refraction are achieved by projecting reflection and refraction maps onto the water surface [AVO02]. A cube map is used to account for global environment (e.g., sky) where the projective texture does not provide pixels of close objects. Finally, we combine those colors in the pixel shader using a Fresnel term.

## 7 Results

We benched our method on a scene containing 35m long 3m wide river. All tests were done on an AMD Athlon 3000+ at 1.8Ghz with an NVIDIA GeForce 8800GTS.

Fig. 16 shows very detailed waves generated by our method in real-time. The wave phenomenon we target is only one of many kinds of waves on river surfaces. Therefore, mixing our model with other wave models is necessary in real applications. Fig. 17 demonstrates that our method is well compatible with other wave models using bump mapping. Fig. 18 shows waves disturbed by floating leaves. Please consult the accompanying video for wave animation and interactive editing of waves <sup>2</sup>.

To demonstrate the performance of our method, we tested it with various view distances (Fig. 19). The simulation time includes two main parts: generating dynamic shockwave crests and constructing wave surface meshes. The result, shown in Table 7, demonstrates that our method is applicable for real-time applications.

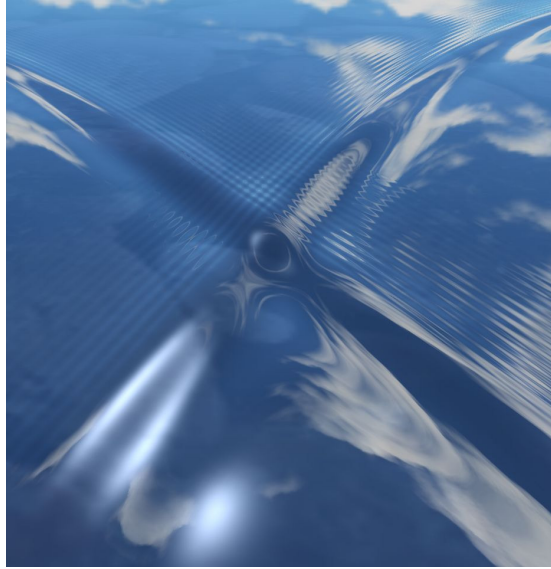
t (ms/frame)	near view	middle view	far view
shockwave crests	2.5	2.5	2.5
wave meshes	12	13	17

**Table 1:** Computational time of our wave simulation method.

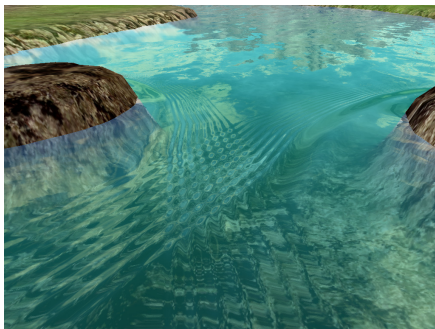
## 8 Conclusion

In this paper, we have proposed a feature-based vector simulation approach for simulating local wave phenomena that often appear in streams. The key idea is to separate

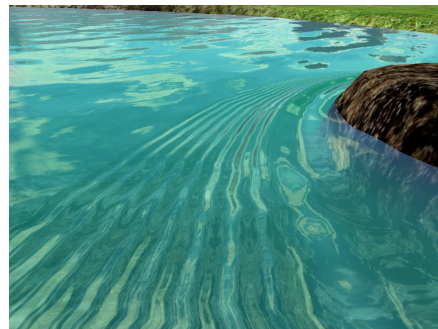
<sup>2</sup>Available at <http://evasion.inrialpes.fr/Membres/Qizhi.Yu/projects/vecsim>



**Figure 16:** *High resolution waves simulated in real-time.*



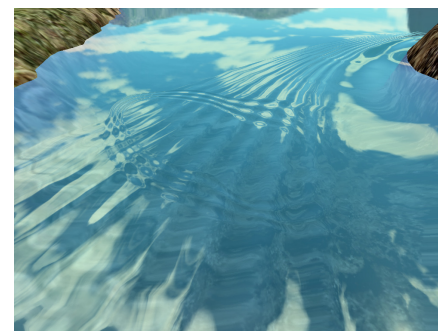
(a)



(b)

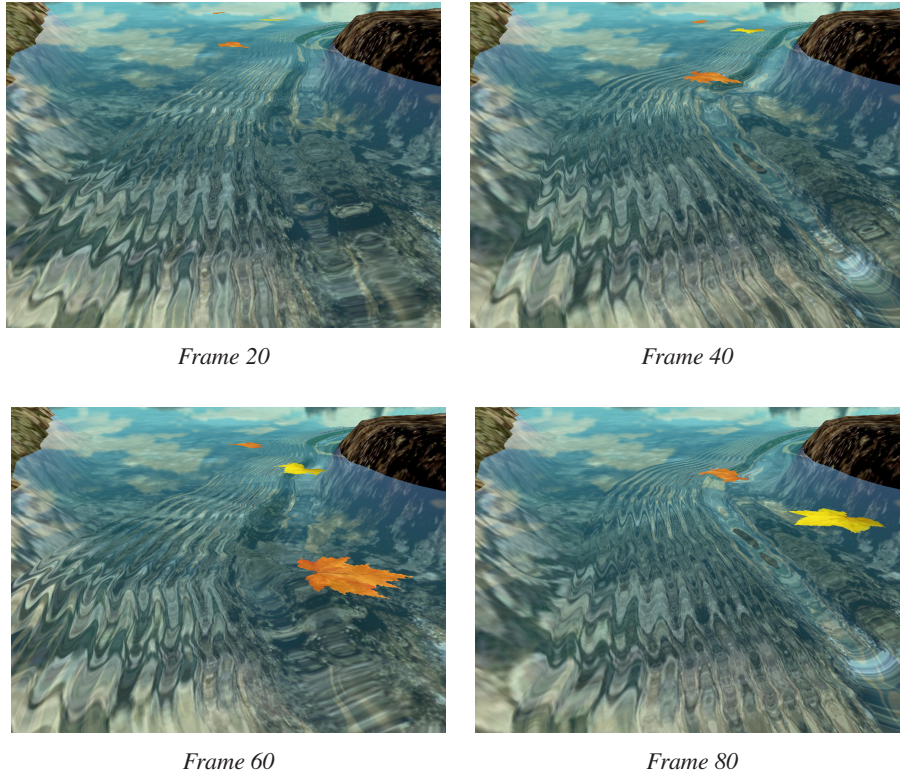


(c)



(d)

**Figure 17:** *By using bump mapping, we can easily combine noise waves (a)(b)(c) and boils (d) with the waves simulated by our method .*



**Figure 18:** Waves disturbed by floating leaves. We get the time varying velocity field by superimposing small local velocity fields attached to leaves on the stationary velocity field.

the representation of the animation and rendering. By using this approach, we have simulated the quasi-stationary waves produced by obstacles. The results show that the vector description allows efficient animation of the waves, and the feature aligned meshes yields high-quality rendering. Further more, the approach allows users to intuitively control the animation. For example, one can easily tune the wave profiles or even keyframe the whole animation.

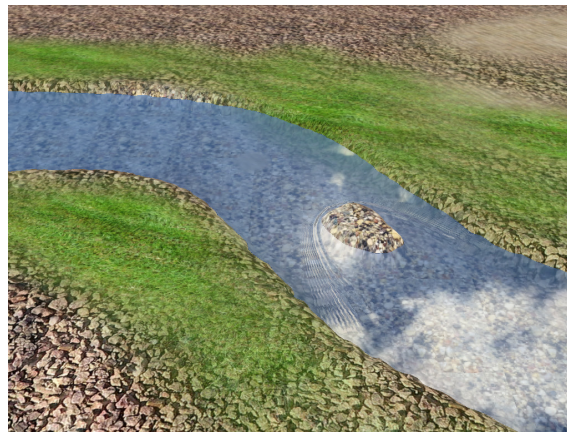
In the future, we would like to apply the presented approach to more local wave phenomena as shown in Fig. 1. For each type of wave, we first need to find an appropriate vector representation and a way to animated it. For this, we can resort to some classical hydrodynamics work as we did in this paper. Next, we need to transform the vector information to some representation that is suitable for high-quality rendering.

**Acknowledgments** The work of the first author was supported by a grant from the European Community under the Marie-Curie project MEST-CT-2004-008270.





(a) Far view.



(b) Middle view.



(c) Close view.

**Figure 19:** Typical views considered in our performance test.

## References

- [AVO02] John Isidoro Alex Vlachos and Chris Oat. Rippling reflective and refractive water. In *ShaderX: Vertex and Pixel shader Programming Tips and Tricks*. Addison-Wesley, 2002.
- [BK01] Mario Botsch and Leif Kobbelt. Resampling feature and blend regions in polygonal meshes for surface anti-aliasing. *Computer Graphics Forum*, 20(3):402–410, September 2001.
- [CdVLHM97] Jim X. Chen, Niels da Vitoria Lobo, Charles E. Hughes, and J. Michael Moshell. Real-time fluid simulation in a dynamic virtual environment. *IEEE Comput. Graph. Appl.*, 17(3):52–61, 1997.
- [EMF02] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 736–744, New York, NY, USA, 2002. ACM Press.
- [FF01] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 23–30, New York, NY, USA, 2001. ACM.
- [FM96] Nick Foster and Demetri Metaxas. Realistic animation of liquids. In Wayne A. Davis and Richard Bartels, editors, *Graphics Interface '96*, pages 204–212. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1996.
- [FM98] Alexey V. Federov and W. Kendal Melville. Nonlinear gravity-capillary waves with forcing and dissipation. *J. Fluid Mech.*, 354:1–42, 1998.
- [FR86] Alain Fournier and William T. Reeves. A simple model of ocean waves. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 75–84, New York, NY, USA, 1986. ACM.
- [Gla02] Andrew Glassner. Duck! *IEEE Computer Graphics and Applications*, 22(4):88–97, 2002.
- [HNC02] Damien Hinsinger, Fabrice Neyret, and Marie-Paule Cani. Interactive animation of ocean waves. In *Symposium on Computer Animation*, july 2002.
- [IGLF06] Geoffrey Irving, Eran Guendelman, Frank Losasso, and Ronald Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 805–811, New York, NY, USA, 2006. ACM.
- [KB00] Leif P. Kobbelt and Mario Botsch. An interactive approach to point cloud triangulation. In *Proc. Eurographics*, 2000.

- [KM90] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 49–57, New York, NY, USA, 1990. ACM.
- [LGF04] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. Simulating water and smoke with an octree data structure. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 457–462, New York, NY, USA, 2004. ACM.
- [LvdP02] Anita T. Layton and Michiel van de Panne. A numerically efficient and stable algorithm for animating water waves. *The Visual Computer*, 18(1):41–53, 2002.
- [Max81] Nelson L. Max. Vectorized procedural models for natural terrain: Waves and islands in the sunset. In *SIGGRAPH '81: Proceedings of the 8th annual conference on Computer graphics and interactive techniques*, pages 317–324, New York, NY, USA, 1981. ACM.
- [MWM87] Gary A. Mastin, Peter A. Watterberg, and John F. Mareda. Fourier synthesis of ocean scenes. *IEEE Comput. Graph. Appl.*, 7(3):16–23, 1987.
- [NP01] Fabrice Neyret and Nathalie Praizelin. Phenomenological simulation of brooks. In *Computer Animation and Simulation*, pages 53–64. Eurographics, Springer, Sep 2001. Eurographics Workshop on Animation and Simulation, Manchester.
- [Pea86] Darwyn R. Peachey. Modeling waves and surf. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 65–74, New York, NY, USA, 1986. ACM.
- [Per85] Ken Perlin. An image synthesizer. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 287–296, New York, NY, USA, 1985. ACM.
- [Sta99] Jos Stam. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.
- [Sto57] J. J. Stoker. *Water Waves: The Mathematical Theory*, volume IV of *Pure and Applied Mathematics*. Interscience Publishers, Inc., 1957.
- [Tes04] Jerry Tessendorf. Simulating ocean water. In *The elements of nature: interactive and realistic techniques*. ACM Press, 2004. SIGGRAPH 2004 Course Notes 31.
- [WH91] Jakub Wejchert and David Haumann. Animation aerodynamics. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 19–22, New York, NY, USA, 1991. ACM.

- [Wu04] Weiming Wu. Depth-averaged two-dimensional numerical modeling of unsteady flow and nonuniform sediment transport in open channels. *Journal of hydraulics Engineering*, 130(10):1013–1024, 2004. CCHE2D free software: <http://www.ncche.olemiss.edu/index.php?page=freesoftware>.
- [YHK07] Cem Yuksel, Donald H. House, and John Keyser. Wave particles. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 99, New York, NY, USA, 2007. ACM.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Related work</b>	<b>4</b>
<b>3</b>	<b>Input data</b>	<b>5</b>
<b>4</b>	<b>Vector simulation</b>	<b>5</b>
4.1	Constructing the contour line of $F_r = 1$ . . . . .	7
4.2	Finding the shockwave starting point on a contour line . . . . .	7
4.3	Tracing wave crest with particles . . . . .	8
4.4	Animating the waves . . . . .	9
<b>5</b>	<b>Efficient high quality rendering of waves</b>	<b>9</b>
5.1	Constructing wave surfaces along shockwave crests . . . . .	10
5.2	Handling the intersection of waves . . . . .	13
<b>6</b>	<b>Implementation details</b>	<b>14</b>
6.1	Wave Profile . . . . .	14
6.2	Bump-mapping and LOD . . . . .	14
6.3	Shading model . . . . .	15
<b>7</b>	<b>Results</b>	<b>15</b>
<b>8</b>	<b>Conclusion</b>	<b>15</b>





---

Centre de recherche INRIA Grenoble – Rhône-Alpes  
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex  
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq  
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique  
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex  
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex  
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex  
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex  
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

---

Éditeur  
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)  
<http://www.inria.fr>  
ISSN 0249-6399