Feature-Based Vector Simulation of Water Waves

Qizhi Yu, Fabrice Neyret, Anthony Steed

▶ **To cite this version:**

Qizhi Yu, Fabrice Neyret, Anthony Steed. Feature-Based Vector Simulation of Water Waves. Computer Animation and Virtual Worlds, 2011, 22 (2-3), pp.91-98. 10.1002/cav.391 . inria-00590401

HAL Id: inria-00590401

https://inria.hal.science/inria-00590401

Submitted on 3 May 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Feature-Based Vector Simulation of Water Waves

Qizhi Yu
LJK/INRIA Grenoble/UCL*

Fabrice Neyret
LJK/INRIA Grenoble

Anthony Steed
UCL

Abstract

We present a method for simulating local water waves caused by obstacles in water streams for real-time graphics applications. Given a low-resolution water surface and velocity field, our method is able to decorate the input water surface with high resolution detail for the animated waves around obstacles. We construct and animate a vector representation of the waves. It is then converted to feature-aligned meshes for capturing the surfaces of the waves. Results demonstrate that our method has the benefits of real-time performance and easy controllability. The method also fits well into a state-of-the-art river animation system.

Keywords: water waves, real-time simulation, procedural methods

1 Introduction

Realistic animation of large bodies of water, such as rivers, lakes or oceans, is an important aspect for graphics applications where natural environments are involved. Though fluid simulation based on solving the full 3D Navier-Stokes equations has achieved great progress in recent years, it is still too computationally expensive to be directly used for the animation of large water volumes. In practice, it is common to simplify the problem to simulating water surface waves based on height-fields.

On the surfaces of water streams, one can easily see some local wave phenomena (Fig. 1) which have quasi-regular and stationary geometric structures. However, they are often missed in current animation applications due to technique difficulties. Using Computational Fluid Dynamics (CFD) techniques, even in 2D [1], would require very high resolution discretization to capture the details of the waves, which is not affordable in real-time applications. Other obstacles to use CFD in this case include complicated physical causes for these local waves, and difficult control of the output waves geometries. Procedural wave models can avoid some of these obstacles, but most of existing ones are designed either for non-local waves [2] or non-flowing water [3] typically for ocean scenes.

In this paper we choose to simulate a typical local wave phenomenon which is caused by obstacles in shallow streams (Figs. 1a and 1b). We approach the problem by locally adding detailed waves to a coarse water surface provided by usual simulation methods. For the local waves, we propose the use of two different representations, vectors and meshes, for simulation and rendering respectively. Our simulation method, improved from Neyret and Praizelin's algorithm [5], uses *vector* representation to capture the geometric *features* of the waves. To achieve high-quality surface details, we propose to build feature-aligned wave meshes. We also propose efficient ways to merge the wave meshes with underlying support water surface and handle wave intersection.

Results show that our method can simulate realistic waves caused by obstacles with very low computational cost. In addition, the geometric shapes of the waves is intuitively controllable. Our method also has good compatibilities with other water wave or water flow models. We envision that the approach of feature-based vector simulation is also applicable for other local wave phenomena.

*This work was mainly done when the first author was a PhD student at LJK/INRIA Grenoble.

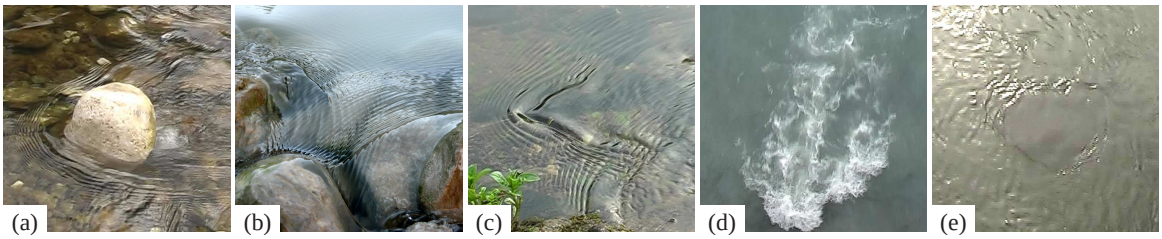

Figure 1: *Photograph of local wave phenomena in water flow: waves caused by obstacles in shallow water and their intersection (a, b), waves caused by underwater obstacles (c), hydraulic jump and foams (d), and boils caused by ejections from river bed [4](e). Please zoom-in for more details.*

2 Related work

In this section we review the water simulation techniques that are suitable for large bodies of water in real-time graphics applications. We refer readers to Bridson's book [6] for a more comprehensive introduction of water simulation for computer graphics.

Procedural techniques are a good choice for cases where computation resources are limited. Perlin noise [7] is a simple way to simulate random water waves. More realistic ocean waves can be achieved by superimposing sinusoidal waves in the spatial domain [8] or spectral domain [2]. Also there exist methods for simulating local waves. Yuksel *et al.* [3] propose a particle system to simulate water waves made by water-object interaction. Glassner [9] simulates ship waves in deep water by geometric construction. Neyret and Praizelin [5] proposed a procedural way to construct the geometric features of the waves caused by obstacles in shallow streams. One of our contributions is to improve it for more stable and efficient animation.

Another category of methods is based on CFD techniques. Water waves in shallow water can be simulated by solving linear wave equations [10] or shallow water equations [11] over a dynamic height field. Kipfer and Westermann [12] simulate river flow with Smoothed Particle Hydrodynamics (SPH) method. These methods can handle interaction better than the procedural ones, but do not scale well. Some real-time CFD-based methods have demonstrated the wave phenomenon we target but only with low resolution surface details [13].

A promising solution for simulating large bodies of water with small scale details in real-time is to couple several models together. Thürey *et al.* [14] add breaking waves upon shallow water simulation. Yu *et al.* [15, 16] simulate rivers by superimposing advected textures on procedurally generated flow. Chentanez and Müler [1] also demonstrate a river animation system that combines the shallow water simulation with other wave models. However, both of the two state-of-the-art river animation systems fail to demonstrate detailed waves caused by obstacles. Our aim is to fill this gap.

3 Input data

We model the surface of a river by superimposing local water waves upon a mean flow (Fig. 2). The mean flow provides not only a support geometry for constructing wave surfaces, but also flow velocity required by our wave simulation algorithm (Section 4). The composition allows that the resolution of the large mean flow could be much coarser than that of the local waves we want to simulate.

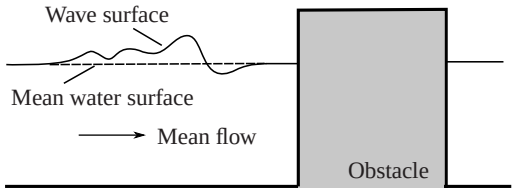

Figure 2: *We model the surface of a river by superimposing local wave surfaces on a mean water surface.*

We assume the mean flow is a steady 2.5D flow as everyday's river is generally calm. Therefore, it has a time-invariant horizontal velocity field $\mathbf{v}_s$, and a time-invariant depth field h. To enhance dynamic details of the flow velocity, we superimpose local perturbations $\mathbf{v}_{p_i}$ on $\mathbf{v}_s$ and get a quasi-stationary flow as in [17]:

$$\mathbf{v} = \mathbf{v}_s + \sum_i \mathbf{v}_{p_i}, \tag{1}$$

where $\mathbf{v}_{p_i}$ could be the analytic solution of sink,

source or vortex with a cutoff radius which is advected with the steady flow velocity $\mathbf{v}_s$.

4 Vector simulation of waves

The purpose of vector simulation is to capture the geometric features of the wave patterns we target (Figs. 1a and 1b). Our method is mostly based on Neyret and Praizelin's algorithm [5] with improvements for better stability and performance. To ease the reader's understanding, we briefly introduce the original algorithm before detailing our improvements to it.

4.1 Original algorithm

Neyret and Praizelin's wave model assumes that the predominant stationary wave caused by an obstacle in a shallow stream can be approximated by the shallow water theory [18, page 22], which suggests that the wave speed $|\mathbf{c}| = \sqrt{gh}$, where g is the gravitational acceleration and h is the local water depth. By using an analogy between the shallow water theory and compressible gas dynamics, the wave in consideration is called *shockwave* in the following discussion. In addition, the model assumes that the shockwave triggers a series of ripples upstream of it (Fig. 3).

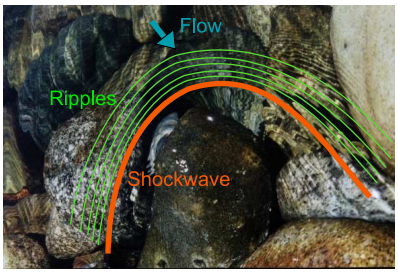

Figure 3: *Neyret and Praizelin's wave model assumes that the waves made by an obstacle consist of a predominant shockwave and a series of ripples.*

Neyret and Praizelin achieved an algorithm for determining the wave crest of the shockwave by analysing its geometric properties (Fig. 4). In a running stream, a wave can be stationary provided its wave speed $|\mathbf{c}| = |\mathbf{v}| \cos\alpha$, where $\mathbf{v}$ is the local flow velocity and α is the propagation angle of the wave relative to the upstream direction. Therefore, the crest of a shockwave should lie at an angle

$$\alpha = \arccos \frac{\sqrt{gh}}{|\mathbf{v}|} \qquad (2)$$

to the upstream direction. At the most upstream point, *starting point*, of a shockwave, the wave crest should be orthogonal to local flow velocity $\mathbf{v}$, *i.e.*, $\alpha = 0$. Substituting it into Eq. 2 gives $|\mathbf{v}| = \sqrt{gh}$ at the starting point. Recall that the Froude number $F_r = \frac{|\mathbf{v}|}{\sqrt{gh}}$. A supercritical flow ($F_r > 1$) past an obstacle leaves a subcritical area ($F_r < 1$) in front of the obstacle. This means the starting point where $F_r = 1$ can be found at the boundary of the subcritical area.

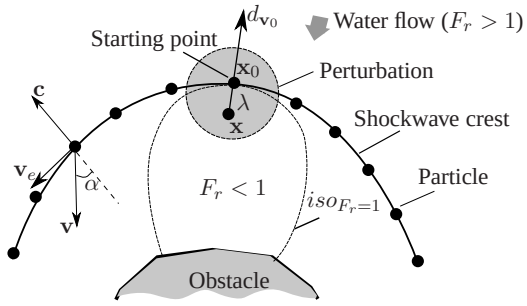

Figure 4: *Schematic illustration of the vector simulation of shockwaves. Our improvements to [5] include: using particles to trace the wave crest, and efficiently finding the starting point.*

Based on the above analysis, Neyret and Praizelin's algorithm at each time step can be summarized as follows:

- Identify a subcritical region upstream from the obstacle by constructing a contour line of $F_r = 1$ starting form the obstacle boundary.
- Find the starting point of the shockwave along the contour line of $F_r = 1$.
- Construct a curve, which represents the shockwave crest, passing through the starting point and satisfying Eq. 2.

In the following sections, we improve the above algorithm for more stable and efficient animation.

4.2 Updating wave starting points

The original algorithm has drawbacks when updating shockwave starting points. At each simulation step, it must reconstruct a new contour line of $F_r = 1$ to locate the new position of the starting point. However, an abrupt velocity perturbation might occasionally yield discontinuous constructions which leads to the popping of wave crests. In addition, reconstructing the contour line at each time step is inefficient.

Considering that the water flow in our problem is quasi-stationary with small perturbations, we precompute the starting point. At every

frame, we directly move the starting point to a nearby position where $F_r = 1$ without constructing a new contour line of $F_r = 1$, thus the new scheme is more robust and efficient.

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. 4, at each time step we locate $\mathbf{x}$ at a distance of λ from $\mathbf{x}_0$ along the direction $d\mathbf{v}_0$:

$$\mathbf{x} = \mathbf{x}_0 + \lambda \frac{d\mathbf{v}_0}{|d\mathbf{v}_0|}, \tag{3}$$

to ensure $|\mathbf{v}(\mathbf{x})| = |\mathbf{c}|$, *i.e.*, $Fr = 1$. Now our task is to solve λ at each time step. Using linear approximation, we estimate the stationary flow velocity with $\mathbf{v}_s(\mathbf{x}) \approx \mathbf{v}_s(\mathbf{x}_0) + \lambda d\mathbf{v}_0$. In addition, we assume that only the nearest perturbation $\mathbf{v}_p$ has a significant influence on the position of the shockwave starting point. Substituting $\mathbf{v}_s(\mathbf{x})$ and $\mathbf{v}_p(\mathbf{x})$ in Eq. 1 yields $\mathbf{v}(\mathbf{x}) = \mathbf{v}_s(\mathbf{x}_0) + \lambda d\mathbf{v}_0 + \mathbf{v}_p(\mathbf{x}_0 + \lambda \frac{d\mathbf{v}_0}{|d\mathbf{v}_0|})$. Then, we can solve $|\mathbf{v}(\mathbf{x})| = |c|$ for λ by using a simple iterative scheme.

4.3 Tracing wave crests with particles

The original algorithm constructs a shockwave crest passing through the starting point by iteratively creating short line segments satisfying Eq. 2. However, this Eulerian scheme makes the propagation of perturbations along the wave crest complicated and unstable. Instead, we propose a Lagrangian particle-based scheme, which is more suitable for advecting physical quantities.

First let's investigate the behavior of the elements of a stationary wave (Fig. 4). 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: $\mathbf{v}_e = \mathbf{v} + \mathbf{c}$. As we discussed in Section 4.1, 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 $|\mathbf{v}_e| = |\mathbf{v}| \sin\alpha = \sqrt{|\mathbf{v}|^2 - |\mathbf{c}|^2}$, and $\mathbf{v}_e$ is tangent to the wave crest.

Based on the above observation, we use particles to represent the wave elements. 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 simulate the attenuation of wave energy, we associate an intensity value to each particle. The intensity value starts from 1 and linearly decreases to 0 with a user given wave attenuation rate k. We kill a particle when its intensity vanishes. Finally, connecting existing particles with line segments gives a shockwave crest.

5 Wave surfaces generation

In this section, we present how to generate high-quality wave surfaces to support realistic rendering in real-time based on the vector simulation.

5.1 Extruding wave surfaces

The waves produced by an obstacle 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 from our vector simulation is only the shockwave crest. 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. 5b). Here, the profile curve represents the superposition of one shockwave and a series of ripples.

With the above assumption, we can 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:

$$\mathbf{S}(u, v) = \mathbf{S}_m(\mathbf{x}(u, v)) + a(u) \cdot z(v) \cdot \mathbf{N}, \tag{4}$$

where S_m is the mean water surface and $\mathbf{x}(u, v) = \mathbf{C}(u) + v \cdot w(u) \cdot \mathbf{B}$. If we neglect the derivative of amplitude $a(u)$, the surface normal can be calculated by

$$\mathbf{N}_w(u, v) = -\frac{a}{w} \frac{\partial z}{\partial v} \mathbf{B} + \mathbf{N}. \tag{5}$$

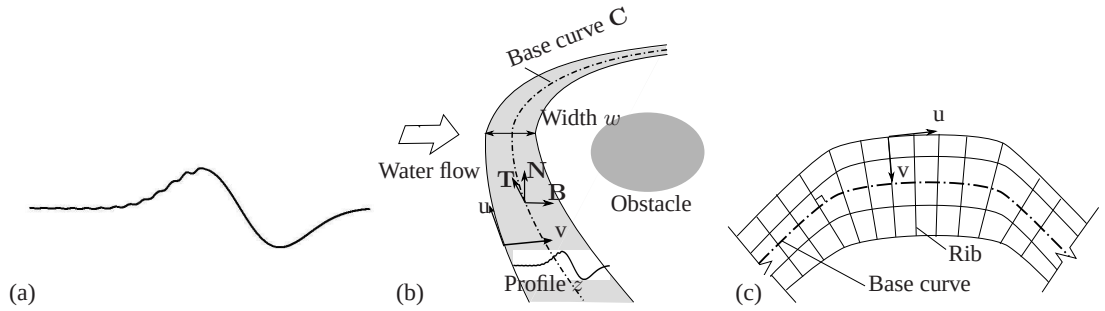

Figure 5: *(a) A custom wave profile $z(v)$ defined by Eq. 6. (b) Sweeping the profile curve along a base curve (a shockwave crest) forms a wave surface. (c) Wave surface is sampled by a quad mesh aligned with iso-parameter lines.*

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

The choice of the wave profile $z(v)$ is up to users except the constraint: $z(v) = z'(v) = z''(v) = 0$ at the two ends. This constraint ensures G^2 continuity between the wave surface and the mean water surface. In our implementation, we used the normalized wave profile illustrated in Fig. 5b. It is defined as the sum of a gravity wave profile $z_g(v)$ and a capillary wave profile $z_c(v)$ inspired by [19]:

$$z(v) = z_g(v) + z_c(v), v \in [-1..1], \quad (6)$$

where $z_g(v) = 2(3v + \frac{1}{\sqrt{2}})e^{-(3v+\frac{1}{\sqrt{2}})^2}$, and $z_c(v) = .045(e^{-2v}\cos(24\pi v) - 1 - v(e^{-2}\cos(24\pi) - 1))$.

5.2 Tessellating wave surfaces

Our task now is to tessellate the parameterized wave surface, $\mathbf{S}(u, v)$, to get a mesh strip that is able to minimize geometric aliasing. As suggested in [20], an effective way to achieve this is to construct a quad mesh whose edges are aligned to iso-parameter lines (Fig. 5c). 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 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 5.5.

5.3 Superimposing waves on mean flow

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. 2). The mesh-stitching technique described in [21] 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 approach the 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 mesh. Still, we need to ensure a perfect continuity between the wave surface and the mean water surface. When the boundary of a wave mesh strip intersects with the edges of the mean water surface mesh in top-down projection, geometry gaps between the two meshes may appear. To avoid this, we split the wave strip at the intersection by inserting extra ribs whose elevation is provided by interpolation on the mean water surface.

5.4 Handling wave intersections

When two obstacles are close, the waves caused by them may intersect and overlap with each other (Fig. 1b). Simply drawing two wave

surfaces without handling the intersection will lead to a non smooth surface, and can not account for the addition of wave amplitudes (Fig. 6a). 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. 6c. The superposed surface of the intersection part is defined by $\mathbf{S}(u_1, u_2, v_1, v_2) = \mathbf{S}_1(u_1, v_1) + \mathbf{S}_2(u_2, v_2) - \mathbf{S}_m(u_1, u_2)$, where S_m is the mean water surface. The surface normal can be calculated by $\mathbf{N}_w(u_1, u_2, v_1, v_2) = -\frac{a_1}{w_1}\frac{\partial z}{\partial v}(v_1) \cdot \mathbf{T}_2 + \frac{a_2}{w_2}\frac{\partial z}{\partial v}(v_2) \cdot \mathbf{T}_1 + \mathbf{B}_1 \times \mathbf{B}_2$. 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 $\mathbf{S}_2$.

5.5 Bump-mapping and LOD

Our wave surface construction allows us to determine accurate per-pixel surface normals through the analytical bump computed from Eq. 5. 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. 6). So, these high frequency waves must be properly filtered. We progressively fade the capillary ripples according to the view distance d. We rewrite Eq. 6 to $z(v) = z_g(v) + \beta z_c(v)$, where β decreases from 1 to 0 as d increases.

To ensure good performance, we determine the resolution of wave mesh strips according to the view distance. We set the level of subdivision in the direction v as $\lfloor \log_2((1-\delta)N_{max}) \rfloor$, with N_{max} the maximum number of grid lines, and $\delta = \frac{d - d_{near}}{d_{far} - d_{near}}$ clamped to $[0, 1]$, where d_{near} and d_{far} correspond to the finest and the coarsest LOD, respectively.

6 Results and discussions

We tested our method in a scene containing a 35 m long 3 m wide river. For input data, we precomputed steady flow velocity and water depth that are defined on a 2D triangle mesh by solving the shallow water equations with the finite volume method [22].

To demonstrate the performance of our method, we tested it with various view distances (Fig. 7). All tests were done on an AMD Athlon 3000+ at 1.8Ghz with an NVIDIA GeForce 8800GTS. The simulation time includes two main parts: generating dynamic shockwave crests and constructing wave surface meshes. The result, shown in Table 1, demonstrates that our method is applicable for real-time applications.

t (ms/frame)	close view	middle view	far view
wave crests	2.5	2.5	2.5
wave meshes	12.0	13.0	17.0

Table 1: *Computational time of our simulation.*

We compare our results with a state-of-the-art real-time SPH-based simulation [13] in Fig. 8. Our result contains much more details of the waves before the obstacle and is more similar to the real waves shown in photograph (Fig. 1).

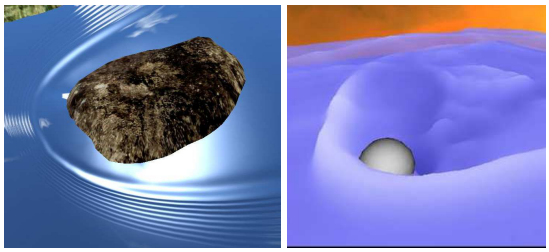

Figure 8: *Comparison between our method (left) and SPH-based simulation (right, reprinted from [13]).*

Fig. 10 shows images from our test on the interaction between the waves and leaves advected with the underlying mean flow. We attach a local analytical source velocity field [17] to each leaf to approximate leaves' perturbations to the mean flow. As one can see in the accompanying video [1], the waves' response to the leaves passing through is visually plausible . Compared with the result of the original algorithm [2], where a whole shockwave sometimes suddenly disappears, our vector simulation is more stable.

Our method provides a set of convenient handles for users to control the output. In the accompanying video, we show an example of user control: the shape of wave patterns can be intuitively modified by tuning the wave width func-

[1] Video available at `http://evasion.inrialpes.fr/Membres/Qizhi.Yu/phd/`.

[2] Video available at `http://www-evasion.imag.fr/Membres/Fabrice.Neyret/brooks/brook.mpg`.

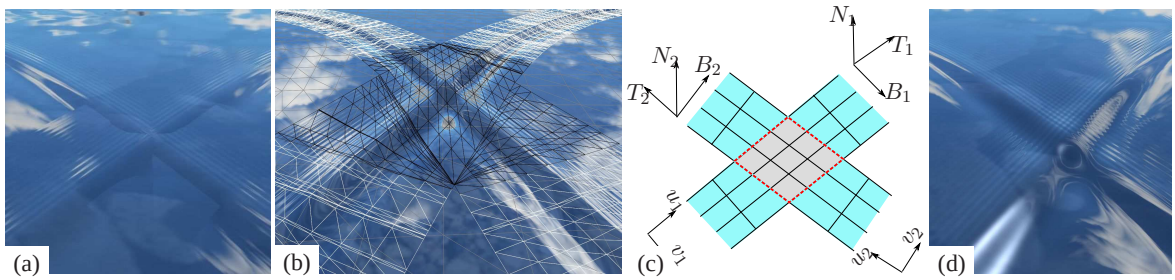

Figure 6: *Handling wave intersections. (a) Simple Z-buffer rendering gives a wrong result. (b) We generate a dedicated mesh patch for the overlapping part. (c) Schematic illustration of our our method. (d) Rendering results of our method.*

Figure 7: *Typical views considered in our performance test: far view (left), middle view (middle), and close view (right)*

tion $w(u)$. Other controls could also be applied through the edit of wave amplitude function $a(u)$, wave profile function $z(v)$, and wave attenuate rate k. We could even apply a factor to the wave velocity $\mathbf{c}$, which will change the curvature of the shockwave crest.

We have demonstrated that our model fits well a practical river animation system that integrates models from different scales [15]. On one hand, in all results our waves are well attached to the input mean flow – large scale water motion. On the other hand, Fig. 9 demonstrates the compatibility with wave models using bump mapping – small scale surface waves.

One limitation of our method is the assumption of gentle water flow. When in a very turbulent flow or under strong perturbations, our 2.5D model may turn to be not reasonable. Another limitation is that our model does not produce wakes behind the obstacle. This could be enhanced by combining procedural flow noise [23] with advected textures [16].

7 Conclusion and future work

In this paper, we have proposed an efficient method for simulating local waves that often appear in shallow and running streams. The key idea is to separate the representation of the waves for animation and rendering, using vectors and meshes respectively. The results show that the vector description allows efficient animation, and the feature aligned meshes ensure high-quality rendering. The proposed method also allows users to intuitionally control the shape of output waves. Finally, our method can be easily incorporated into state-of-the-art hybrid river animation systems.

One of our future work is to avoid the precomputation of the input data. The river flow could be generated on-the-fly by using a procedural method [15], or a fast shallow water solver [1].

The approach of simulating vector features of the waves would connect a wide range of previous studies on water waves to computer animation. We could utilize the results of structural analysis [24] or experimental measurements [4] of local water waves to simulate more wave phenomena shown in Fig. 1.

Acknowledgements

The first author was supported by an EU Marie Curie fellowship during this work.

References

[1] Nuttapong Chentanez and Matthias Müler. Real-time simulation of large bodies of water with small scale details. In *Proc. Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, 2010.

[2] Jerry Tessendorf. Simulating ocean water. In *The elements of nature: interactive and realistic techniques*. 2004. SIGGRAPH 2004 Course Notes 31.

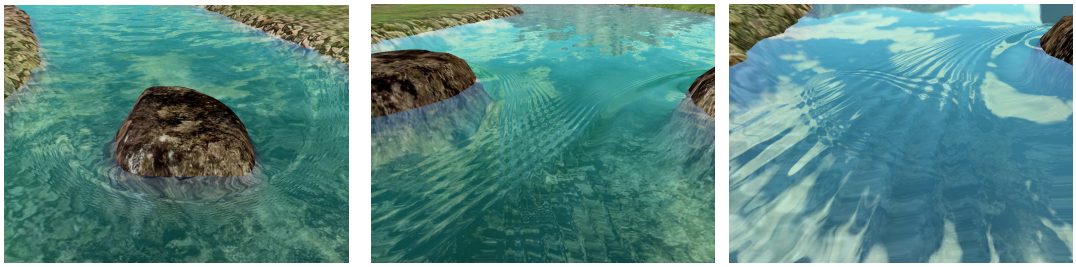

Figure 9: *Integration with other waves: noise waves (left and middle), and boils (right).*

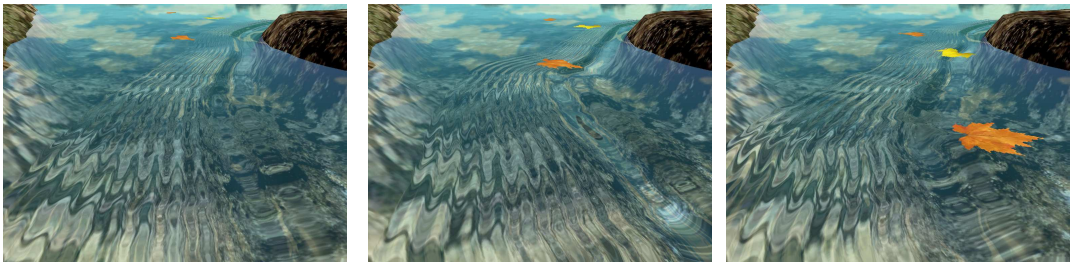

Figure 10: *Interaction between floating leaves and waves.*

[3] Cem Yuksel, Donald H. House, and John Keyser. Wave particles. *ACM Trans. Graph.*, 26(3):99, 2007.

[4] Roscoe G. Jackson. Sedimentological and fluid-dynamic implications of the turbulent bursting phenomenon in geophysical flows. *Journal of Fluid Mechanics*, 77:531–560, 1976.

[5] Fabrice Neyret and Nathalie Praizelin. Phenomenological simulation of brooks. In *Proc. Eurographic Workshop on Computer Animation and Simulation*, pages 53–64, 2001.

[6] Robert Bridson. *Fluid Simulation for Computer Graphics*. A K Peters, 2008.

[7] Ken Perlin. An image synthesizer. In *Proc. SIGGRAPH '85*, pages 287–296, 1985.

[8] Alain Fournier and William T. Reeves. A simple model of ocean waves. In *Proc. SIGGRAPH '86*, pages 75–84, 1986.

[9] Andrew Glassner. Duck! *IEEE Computer Graphics and Applications*, 22(4):88–97, 2002.

[10] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *Proc. SIGGRAPH '90*, pages 49–57, 1990.

[11] 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.

[12] Peter Kipfer and Rüdiger Westermann. Realistic and interactive simulation of rivers. In *Proc. Graphics Interface*, pages 41–48, 2006.

[13] Hyokwang Lee and Soonhung Han. Solving the shallow water equations using 2D SPH particles for interactive applications. *The Visual Computer*, 26:865–872, 2010.

[14] Nils Thürey, Matthias Müller-Fischer, Simon Schirm, and Markus Gross. Real-time breaking waves for shallow water simulations. In *Proc. Pacific Conference on Computer Graphics and Applications*, pages 39–46, 2007.

[15] Qizhi Yu, Fabrice Neyret, Eric Bruneton, and Nicolas Holzschuch. Scalable real-time animation of rivers. *Proc. Eurographics 2009*, 28(2):239–248, 2009.

[16] Qizhi Yu, Fabrice Neyret, Eric Bruneton, and Nicolas Holzschuch. Lagrangian texture advection: Preserving both spectrum and velocity field. *IEEE Transactions on Visualization and Computer Graphics*, 99(PrePrints), 2010.

[17] Jakub Wejchert and David Haumann. Animation aerodynamics. In *Proc. SIGGRAPH '91*, pages 19–22, 1991.

[18] J. J. Stoker. *Water Waves: The Mathematical Theory*, volume IV of *Pure and Applied Mathematics*. Interscience Publishers, Inc., 1957.

[19] Alexey V. Federov and W. Kendal Melville. Nonlinear gravity-capillary waves with forcing and dissipation. *J. Fluid Mech.*, 354:1–42, 1998.

[20] Mario Botsch and Leif Kobbelt. Resampling feature and blend regions in polygonal meshes for surface anti-aliasing. *Computer Graphics Forum*, 20(3):402–410, 2001.

[21] Leif P. Kobbelt and Mario Botsch. An interactive approach to point cloud triangulation. *Computer Graphics Forum*, 19(3):479–487, 2000.

[22] 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.

[23] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. *ACM Trans. Graph.*, 26, 2007.

[24] Vallé B. L. and G. B. Pasternack. Air concentrations of submerged and unsubmerged hydraulic jumps in a bedrock step-pool channel. *Journal of Geophysical Research*, 111:12, 2006.