Real-time Animation and Rendering of Ocean Whitecaps

Jonathan Dupuy * Eric Bruneton †

INRIA Grenoble Rhône-Alpes, Université de Grenoble et CNRS, Laboratoire Jean Kuntzmann
Université de Lyon, CNRS Université Lyon 1, LIRIS, UMR5205, F-69622, France

Figure 1: *Some real-time results obtained with our method showing large ocean scenes with whitecaps under different wave and illumination conditions. The whitecap contribution is correctly averaged as the viewing distance increases.*

Abstract

We present a scalable method to procedurally animate and render vast ocean scenes with whitecaps on the GPU. The whitecap coverage on the ocean surface is determined using a wave deformation criteria which can be pre-filtered linearly. This allows us to take advantage of the fast mip-mapping and texture filtering capabilities of modern hardware and produce plausible and anti-aliased images for scales ranging from centimetric to planetary in real time.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Antialiasing I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture—;

Keywords: whitecaps, ocean, BRDF, real-time

Links: DL PDF

1 Introduction

Oceans are highly dynamic structures: their surface is constantly perturbed through time by waves of different directions, wavelengths and amplitudes. When conditions prevail [Cokelet 1977], some of these waves break, causing very perceptible reflectance changes –called whitecaps– and geometrical alterations on the ocean surface. In digital environments, these phenomenons can be reproduced through complex mathematical models, which often require a lot of processing power.

Whenever applications can not afford high simulation times, such as video games, only the models meeting the computational constraints can be employed. In general, these models either completely ignore whitecaps or use very crude empirical approximations [Premože and Ashikhmin 2001; Darles et al. 2007]. Recent methods can offer more plausible results through the use of particle systems [Chentanez and Müller 2011]; however, such solutions do not scale well enough to handle vast environments. Hence a huge gap exists between the quality of ocean scenes produced by real-time renderers and offline simulators.

In this work, we present a procedural whitecap generation algorithm which narrows down this gap. Our method runs entirely on the GPU and allows us to render high quality ocean scenes in a few milliseconds. In the two following sections, we show how to produce a scalable ocean surface and derive a whitecap model with linear parameters, which in turn can be evaluated for any viewing resolution by the hardware. We then give some implementation details and conclude our paper in the fifth section.

2 Ocean Model

Wave Model

The core of our wave model is based on the Choppy Wave Model [Tessendorf 2001]: At a given time, the vertical and horizontal perturbations produced by the ocean waves can be approximated by sampling an empirical spectrum in frequency space. Because the sampling process can be done through a Fast Fourier Transform, the resulting deformations can be bounded to a small area and tiled seamlessly on the ocean surface.

Tiling a single perturbation pattern however leads to repetitive artifacts or lack of detail depending on the observation scale. Previous work solves this by generating additional noise on the surface [Rydahl 2009; Nvidia 2011]. Instead, we chose to compute a number $n \in \mathbb{N}^{*+}$ of patterns, each sampling a different part of the wave spectrum. This approach does not remove the periodicity of the final wave field, but rather increases it to the least common multiple

*e-mail:jdupuy@liris.cnrs.fr
†e-mail:ebruneton@free.fr

of the n pattern periods (in practice, $n = 4$ is enough to avoid artifacts). It also provides an analytical expression for the whole ocean surface: Given a point $\mathbf{p}(p_x, p_y, p_z)$ located on the ocean plane at rest, its position $\mathbf{q}(q_x, q_y, q_z)$ at time t is given by

$$\mathbf{q}(\mathbf{p}, t) = \mathbf{p} + \sum_{i=0}^{n-1} \begin{bmatrix} \lambda_i u_i(\mathbf{p}, t) \\ \lambda_i v_i(\mathbf{p}, t) \\ w_i(\mathbf{p}, t) \end{bmatrix}, \quad \lambda_i \in \mathbb{R}^+ \tag{1}$$

where $w_i(\mathbf{p}, t)$ (resp. $\mathbf{D}_i(\mathbf{p}, t) = [u_i(\mathbf{p}, t),\, v_i(\mathbf{p}, t)]^{\mathrm{T}}$) is the choppy wave model's vertical (resp. horizontal) deformation function for a fixed pattern size [Tessendorf 2001], and λ_i is an artistic parameter which controls the choppiness of the waves.

Surface Mesh

In order to represent the ocean surface efficiently, a multi-resolution meshing scheme has to be employed. We chose to use the projected grid [Hinsinger et al. 2002]: a mesh is generated in screen space and reprojected on the ocean plane in the world coordinate system. Each vertex is then displaced by evaluating (1) and reprojected to the screen. This approach naturally provides adaptive level of detail with constant memory consumption, which is very convenient for GPU-based renderers.

Surface Radiance

According to Zhang et al. [Zhang et al. 2006], the ocean surface radiance R contributing to a pixel can be modeled by

$$R \simeq W R_F + R_C + R_W \tag{2}$$

where $W \in [0, 1]$ is the whitecap fractional coverage and R_F, R_C, and R_W respectively denote the whitecap foam, clear (foam free), and water-leaving radiance. Because our ocean wave model is analytical, we can use the work of Bruneton et al. [Bruneton et al. 2010] to compute R_C and R_W. Similarly, we follow Premože et al. [Premože and Ashikhmin 2001] and set R_F to a smooth Lambertian surface with a reflectance of 0.4, a reasonnable approximation in the visible spectrum range [Koepke 1984].

With these assumptions, the only parameter left to evaluate is W, which is addressed in the next section.

3 Whitecap Model

Discrete Generation

Most of the whitecaps appearing on the ocean surface are due to wave breaking [Anguelova and Webster 2006]. We detect such events by evaluating the choppiness of the surface at each point $\mathbf{p}$ as suggested by Tessendorf [Tessendorf 2001], which is inversely proportional to the Jacobian determinant of the horizontal transformations, denoted by $j(\mathbf{p}, t)$. We chose to compute the following function

$$b(\mathbf{p}, t) = \Upsilon\left(\epsilon - j(\mathbf{p}, t)\right), \quad \epsilon \in \mathbb{R} \tag{3}$$

and generate whitecaps whenever $b(\mathbf{p}, t) = 1$, where Υ is the Heaviside function and

$$j(\mathbf{p}, t) = \begin{vmatrix} 1 + \sum_i \lambda_i \frac{\partial u_i(\mathbf{p},t)}{\partial x} & \sum_i \lambda_i \frac{\partial u_i(\mathbf{p},t)}{\partial y} \\ \sum_i \lambda_i \frac{\partial u_i(\mathbf{p},t)}{\partial y} & 1 + \sum_i \lambda_i \frac{\partial v_i(\mathbf{p},t)}{\partial y} \end{vmatrix} \tag{4}$$

The variable ϵ controls the amount of breaking. Note that this scheme does not reproduce the geometric deformations produced by wave breaking, but may be used to model the changes of reflectance it causes on the surface.

Per Pixel Filtering

If we neglect the shadowing and masking produced by the waves, the whitecap fractional coverage W using our wave breaking model inside a pixel is given by

$$W = \frac{1}{A} \iint_A b(\mathbf{p}, t)\, \mathrm{d}\mathbf{p} \tag{5}$$

where A bounds the footprint of the pixel on the ocean plane. This integral is clearly not suitable for real-time renderers as it is not linear, nor can it be solved analytically. Therefore, approximations have to be made in order to reduce its complexity.

For large footprints, it has been observed that wave slopes tend to be normally distributed on ocean surfaces [Ross et al. 2005]. Since our formulation of j directly builds on these features, we assume that it also follows a normal distribution. This allows to us to replace (5) by an integration over a probability distribution function, thus obtaining (dropping dependencies on $\mathbf{p}$ and t)

$$\begin{aligned} W &\approx \int_{-\infty}^{+\infty} \Upsilon(\epsilon - j) \frac{1}{\sqrt{2\pi\sigma_A^2}} \exp\left(-\frac{(j - \mu_A)^2}{2\sigma_A^2}\right) \mathrm{d}j \\ &= \frac{1}{2} + \frac{1}{2}\mathrm{erf}\left(\frac{\sqrt{2}}{2\sigma_A}(\epsilon - \mu_A)\right) \end{aligned} \tag{6}$$

where erf, μ_A and σ_A^2 respectively denote the error function, the mean, and the variance of j in A. Now, W is an analytic expression whose parameters μ_A and σ_A^2 can be evaluated using fast linear pre-filtering methods [Bruneton and Neyret 2011] such as the hardware accelerated mipmap generator available on modern GPUs.

Storage

Linear pre-filtering methods require the (linear) data to be stored in textures or arrays. Since the spatial period of $\mathbf{q}(\mathbf{p}, t)$ can be very large, pre-filtering μ_A and σ_A^2 in (6) may require a prohibitive amount of memory. We show however that these variables can be evaluated using much less memory by reformulating them as sums of contributions from each wave pattern.

Developping expression (4) for any number $n \in \mathbb{N}^{*+}$ of patterns yields

$$j = \sum_{i=0}^{n-1} \{\frac{1}{n} + a_i + b_i + a_i b_i - c_i^2\} + \sum_{i \neq j} \{a_i b_j - c_i c_j\}$$

where $a_i = \lambda_i \frac{\partial u_i}{\partial x}$, $b_i = \lambda_i \frac{\partial v_i}{\partial y}$, $c_i = \lambda_i \frac{\partial u_i}{\partial y}$. Each term in the first sum only depends on one pattern and can easily be stored in a small linearly pre-filterable texture. The terms in the second sum involve products of uncorrelated functions, and thus their average is zero. Therefore we get

$$\mu_A = \frac{1}{A} \sum_i \iint_A k_i\, \mathrm{d}\mathbf{p}, \quad k_i = \frac{1}{n} + a_i + b_i + a_i b_i - c_i^2 \tag{7}$$

Similarly, developping the square of (4) gives $\sum_i k_i^2$ plus terms which involve products of uncorrelated functions, whose average is also zero, and thus we have

$$\sigma_A^2 = -\mu_A^2 + \frac{1}{A} \sum_i \iint_A k_i^2\, \mathrm{d}\mathbf{p} \tag{8}$$

4 Implementation and Results

We implemented our method in C++ using OpenGL and GLSL, which runs in two main passes. The first pass evaluates the terms of equations (1) (4) as well as the k_i and k_i^2 factors from (7) and (8) and stores the results in 2D textures with mipmaps. Then, we render the projected grid as described in Section 1 in a vertex shader and evaluate equation (2) in a fragment shader. Since the terms of W are stored in textures, the GPU can automatically select the correct mip and anisotropy levels for any viewpoint. Results and performances are respectively shown in Figures 1 and 2.

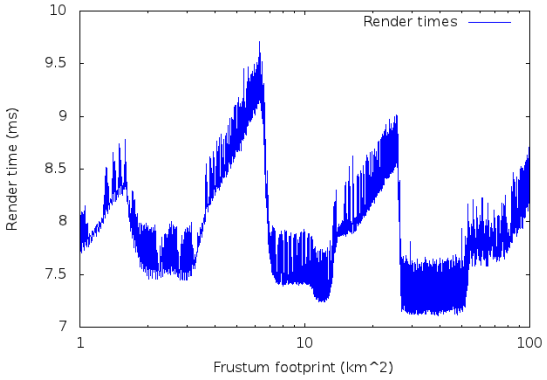

Figure 2: *Rendering times depending on the footprint of the camera. A footprint of* $100km^2$ *corresponds to an altitude of* $4.6km$ *if the camera is pointing straight down. Frames are generated in less than* $10ms$ *at* 1280×720 *resolution on an NVIDIA GeForce* $560Ti$*.*

Using four different patterns requires a total of eight distinct 2D RGBA textures for storage, which consumes up to 5.33 MB (accounting for the whole mip pyramid) of memory at half floating point precision. The complete source code is available on github at `http://github.com/jdupuy/whitecaps`

5 Conclusion and Future Work

We have presented a new model to explore ocean scenes from ground to space with whitecaps in real time. Our implementation runs entirely on the GPU and takes full advantage of available hardware features to integrate the optical effects occuring at the ocean surface.

A lot of directions can be taken for future work:

- First, our model only produces whitecaps whenever a wave breaks, but does not handle their decay which can last up to several seconds, also causing reflectance fluctuations [Koepke 1984].
- Second, equation (5) does not take masking effects into account, which is important at grazing angles. This could be solved by using the masking function of Ross et al. [Ross et al. 2005], already used in our framework to evaluate the sun's reflected radiance in R_C in equation (2).
- Lastly, the lambertian properties of the foam surface could be replaced by a richer shading model.

Acknowledgements

The authors thank Fabrice Neyret for his advice in the early stage of the project and Jean-Claude Iehl and Pierre Poulin for proofreading the article. This work was cunducted while both authors were at INRIA Grenoble Rhône-Alpes and was funded by the ANR 2010 JCJC 0207 01 "SimOne" project.

References

ANGUELOVA, M., AND WEBSTER, F. 2006. Whitecap coverage from satellite measurements: A first step toward modeling the variability of oceanic whitecaps. *J. Geophys. Res. 111*.

BRUNETON, E., AND NEYRET, F. 2011. A Survey of Non-linear Pre-filtering Methods for Efficient and Accurate Surface Shading. *IEEE Transactions on Visualization and Computer Graphics*.

BRUNETON, E., NEYRET, F., AND HOLZSCHUCH, N. 2010. Real-time Realistic Ocean Lighting using Seamless Transitions from Geometry to BRDF. *Computer Graphics Forum 29*, 487–496.

CHENTANEZ, N., AND MÜLLER, M. 2011. Real-time Eulerian Water Simulation Using a Restricted Tall Cell Grid. *ACM Trans. Graph. 30* (August), 1–10.

COKELET, E. D. 1977. Breaking waves. *Nature 267*, 769–774.

DARLES, E., CRESPIN, B., AND GHAZANFARPOUR, D. 2007. Accelerating and Enhancing Rendering of Realistic Ocean Scenes. *WSCG*.

HINSINGER, D., NEYRET, F., AND CANI, M.-P. 2002. Interactive Animation of Ocean Waves. In *ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA)*.

KOEPKE, P. 1984. Effective reflectance of oceanic whitecaps. *Appl. Opt. 23*, 1816–1824.

NVIDIA. 2011. Ocean Surface Simulation. *NVIDIA Graphics SDK 11 Direct3D*.

PREMOŽE, S., AND ASHIKHMIN, M. 2001. Rendering Natural Waters. *Computer Graphics forum 20*, 4, 189–199.

ROSS, V., DION, D., AND POTVIN, G. 2005. Detailed analytical approach to the Gaussian surface bidirectional reflectance distribution function specular component applied to the sea surface. *JOSA A 22*, 11, 2442–2453.

RYDAHL, B. 2009. A VFX ocean toolkit with real time preview. *Student thesis*.

TESSENDORF, J. 2001. Simulating Ocean Water. *ACM SIGGRAPH course notes*.

ZHANG, X., HE, M.-X., YANG, Q., AND ZENG, K. 2006. Effects of Winds on Ocean Color. *Geoscience and Remote Sensing Symposium*, 4056–4059.