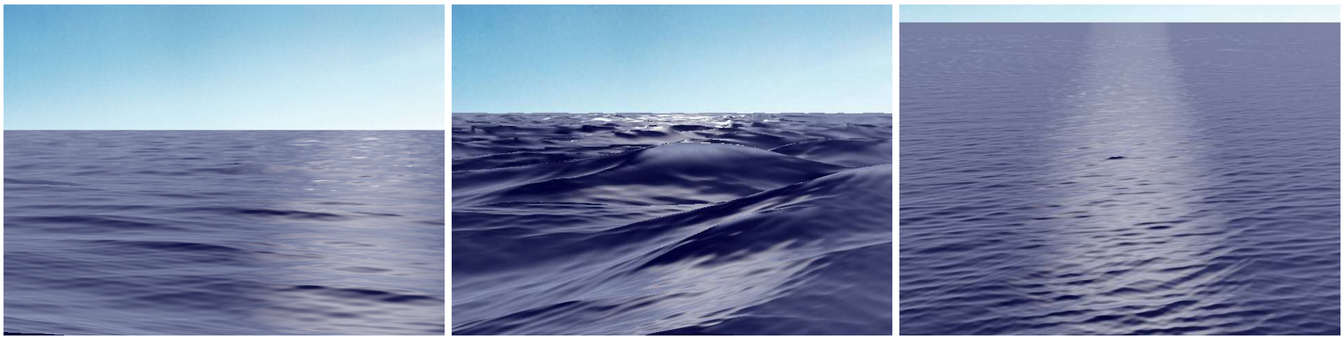
Interactive Animation of Ocean Waves

Damien Hinsinger
Damien.Hinsinger@club-internet.fr

Fabrice Neyret
Fabrice.Neyret@imag.fr

Marie-Paule Cani
Marie-Paule.Cani@imag.fr

iMAGIS-GRAVIR, joint research project of CNRS,INPG,INRIA,UJF



Abstract

We present an adaptive scheme for the interactive animation and display of ocean waves far from the coast. Relying on a procedural wave model, the method restricts computations to the visible part of the ocean surface, adapts the geometric resolution to the viewing distance and only considers the visible waves wavelengths. This yields real-time performances, even when the camera moves. The method allows the user to interactively fly over an unbounded animated ocean, which was not possible using previous approaches.

Keywords: natural phenomena, procedural animation, adaptive scheme, ocean, waves

1 Introduction

Developing interactive models for natural phenomena is even more important now that it was in the past: People like to be immersed in rich and open spaces. Games thus start providing more nature and exterior scenes, following the special effect and CG film industries. However, providing nice looking natural elements such as forests, clouds or animated water at interactive rates is difficult. Two common problems of natural sceneries is that they are both very wide and very detailed. This is especially true for the ocean surface, since waves exist at several scales and extend up to the horizon. Most of the recent ocean waves animations featured in movies were created using a spectral approach [Areté Image Software n. d.]. The latter restrict computations to a rectangular area, which can be used as a tile for animating a larger surface. The computation cost only allows crude resolution when interactive rates are required. Moreover, enabling the user to interactively fly over the ocean is impossible, since avoiding obvious repetitivity would require the use of an extremely large simulated region.

This paper presents an adaptive scheme for the interactive animation and display of an unbounded ocean. It relies on a procedural wave model which expresses surface point displacements as sums of contributions from the active wave-trains, modeled as independent animation primitives. This model allows to concentrate computations on the part of the ocean surface and on the wavelengths that are currently visible. Our main contribution is a scheme for dynamically adapting both the geometric resolution and the set of animation primitives: surface sampling is adapted to the current viewpoint, while wave trains that do not correspond to visible frequencies are filtered. This yields interactive rates, even when the camera moves under the user control. In addition to enabling interactive fly-over, the method eases the addition of extra local animation primitives such as ship waves.

The remainder of this paper develops as follows: Section 2 reviews the existing models for simulating ocean waves. Then we present our procedural wave model in Section 3. The adaptive scheme is detailed in Section 4. We show and discuss results in Section 5, and conclude in Section 6.

2 Previous Work

2.1 Wave theory

Wave generation and frequency

Ordinary ocean waves are created by the wind in *fetch* areas, and can propagate far from these locations. Several models have been proposed to account for the amplitude, frequency and direction spectrum according to the wind strength and duration [Kinsman 1984]. A classical one is the Pierson-Moskowitz filter giving the amplitude F_{PM} in function of the frequency f :

$F_{PM}(f) = \frac{ag^2}{(2\pi)^4 f^5} e^{\frac{5}{4}\left(\frac{f_m}{f}\right)^4}$, where g is the gravity acceleration, a is the Phillips constant, and $f_m = \frac{0.13g}{U_{10}}$ corresponds to the peak in the spectrum (which has a Gaussian-like shape), depending on the wind velocity at a 10*m* altitude. This 1D filter has been extended in 2D by Hasselmann et al. [Hasselmann et al. 1980] using the formulation $F(f,\alpha) = F_{PM}(f)D(f,\alpha)$, where $D(f,\alpha)$ weights the filter according to the propagation direction α.

Velocity and wavelength

In contrast to sound waves in air, surface water waves are *dispersive*, i.e. their velocity depends on their wavelength. For a wave of pulsation $\omega = \frac{2\pi}{T}$ and wave number $k = \frac{2\pi}{\lambda}$ (where T is the period and λ is the wavelength), the relation $\omega^2 = (gk + \frac{\gamma}{\rho}k^3)\tanh(kH)$ (where g is the gravity acceleration, ρ the water density, H the water depth, and γ controls the surface tension) links the two variables (see [Lighthill 1978; Crawford 1977]). The first term corresponds to ordinary gravity waves, and the second term corresponds to capillary waves. As the latter only gives wavelengths lower than 5.4 *mm*, they are usually neglected for ocean simulation. The velocity is thus $c = \frac{\omega}{k} = \sqrt{\frac{g}{k} tanh(kH)}$. The $\tanh(kH)$ factor can be neglected in the case of deep water. Otherwise, the wavelength and the velocity of waves will change with water depth, both decreasing when approaching the shore.

Waves shape

Several models characterize the shape of waves by studying eigenmodes of the Navier-Stokes equation at the water-air interface (see [Lighthill 1978; Kinsman 1984]). A convenient one is the Gerstner swell model, which describes the trajectory of water particles as circles of radius equal to the wave amplitude A around the location at rest. Two particles along the direction of wave propagation having a distance at rest of l follow their circular trajectories at angular velocity ω with a phase difference of kl:

$$\begin{cases} x - x_0 = Ae^{kz_0}\sin(\omega t - kx_0) \\ z - z_0 = Ae^{kz_0}\cos(\omega t - kx_0) \end{cases}$$

where t is the time, z the vertical axis and (x_0, z_0) the particle location at rest. This generates a *trochoid* wave shape, similar to a sinusoid only for very small amplitudes. For high amplitudes the waves get choppy, up to a value for which the curve crosses itself, which is no longer physical since the wave should break.

Other models, like Stokes and Biesel's ones, take into account the shallow water case, for which the circles turn into ellipses, at the price of more complicated formulas. Since the depth variation changes the wave velocity, the phases are no longer linear with the distance. Biesel's model thus evaluates the phases as $\int_0^{x_0} k(x)dx$. This change is responsible for the refraction of wave trains[1] close to the shore.

2.2 Computer Graphics models

We do not review here papers dedicated to running water or to rivers such as [Kass and Miller 1990; Foster and Metaxas 1996; Foster and Fedkiw 2001; Neyret and Praizelin 2001; Thon and Ghazanfarpour 2001], which focus on other kinds of water surfaces and are not adapted to the simulation of ocean waves.

To date, several ocean wave models have been developed in CG, which can be separated in two families:

[1] A wave train is a (potentially infinite) set of similar waves whose fronts are nearly parallels and evenly spaced.

The first one starts with the work of Fournier and Reeves [Fournier and Reeves 1986] that simulate a train of trochoids, relying on a mix of Gerstner and Biesel swell models. Since they solve the direction of propagation and the phases of the trains on the surface, they have to regularly mesh the simulated region of ocean surface. The same year, Peachey [Peachey 1986] proposed a similar idea, with fewer refinements (e.g., no trochoids). More recently, several models [Ts'o and Barsky 1987; Gonzato and Saëc 2000] have proposed more precise ways to solve the propagation (*wave tracing*). Note that noise is generally used in all the models above in order to avoid the visual regularity due to the fact that only one or two wave trains are simulated.

The second family corresponds to spectral approaches, first introduced in CG by Mastin et al. [Mastin et al. 1987]. The basic idea is to produce an height field having the same spectrum as the ocean surface. This can be done by filtering a white noise with Pierson-Moskowitz's or Hasselmann's filter, and then calculating its Fast Fourier Transform (FFT). The main benefits of this approach are that many different waves are simultaneously simulated, with visually pleasing results. However, animating the resulting ocean surface remains challenging. Moreover, the method generates sine waves, which only approximate trochoids in the calm sea case. Tessendorf [Tessendorf 1999] shows that dispersive propagation can be managed in the frequency domain and that the resulting field can be modified to yield trochoid waves. A positive property of FFTs is the cyclicity: the solution can be used as a tile, which allows to enlarge the usable surface region as long as the repetitiveness is not obvious. The corresponding negative aspect of FFTs is homogeneity: no local property can exist, so no refraction can be handled. It should be noted that this model is the one implemented by Areté Image Software [Areté Image Software n. d.] and used for the special effects of many movies such as *Waterworld* or *Titanic*.

Thon et al. [Thon et al. 2000] use an hybrid approach: the spectrum synthesized using a spectral approach is used to control the trochoids generated by a Gerstner model. This is only applicable in the calm sea case, where trochoids of small amplitude are very similar to sines. Smaller scale waves are obtained by directly tuning some extra Perlin noise [Perlin 1985].

For both families of approaches, a pre-defined region of the ocean surface is simulated and regularly tessellated, whatever the camera position. Since the visible surface area is large and a high resolution is needed in regions close to the camera, this requires very large meshes, yielding costly simulations. These methods can only be adapted to real-time for very crude resolutions.

Recently, **adaptive schemes** have successfully been used for efficient modeling, rendering or animation of complex objects [Stollnitz et al. n. d.; Pfister et al. 2000; Debunne et al. 2001]. The idea is to minimize the sampling of the geometry according to criteria such as the local variation amplitude of the parameters of interest, or distance from the viewpoint. In particular, *surfels* techniques [Pfister et al. 2000] directly use discs or ellipses to sample the geometry. In [Stamminger and Drettakis 2001] the number of samples is proportional to the size of the screen projection of the objects. Since the adaptive sampling is done on the fly for each frame, this fits well with procedural surface displacement, which can easily be animated. In the same spirit, we rely on an adaptive sampling of the ocean surface, dictated by the camera position. Moreover, our animation model is also adaptive, since we filter the wave trains that cannot be observed from the current viewpoint. These adaptive schemes are made possible by the use of a procedural wave model, described next.

3 Wave Model

We are looking for a wave model that does not constrain us to simulate a predetermined and regularly sampled surface region. Moreover, we are not willing to compute a hight field, which would not cover the case of stormy seas. We keep using a mesh (since we cannot afford pixel size elements), but its location in world space will change dynamically and its density varies in space. This leads us to the use of a procedural wave model, following the first family of CG approaches mentioned above.

Our model is based on the Gerstner swell model and simulates trochoids. However, we want to take into account the combination of many different waves. To do so, we generate wave trains in a way that approaches a known wave spectrum, in the same spirit as Thon et al. [Thon et al. 2000]. Note that our model is dedicated to deep waters, and thus does not cover waves refraction near shore nor wave breaking.

3.1 Wave trains

Although our method can be applied to various kinds of waves (an experiment with ship waves will be shown in the Section 5), this paper only focuses on the main ones, i.e. gravity waves.

We simulate gravity waves using a series of wave trains that homogeneously cover the world. Their amplitude a_i, frequency $\frac{\omega_i}{2\pi}$ and direction $\frac{K_i}{|K_i|}$ are chosen to reproduce a given reference spectrum[2]. They are chosen manually in our implementation, although an automatic choice such as the one in [Thon et al. 2000] could be used as well.

3.2 Animation of mesh points

Let us consider the mesh that represents the ocean surface at a given animation step (the way this mesh is generated from the current camera position will be explained in Section 4).

The mesh vertices are considered as particles, and thus follow the circle trajectory corresponding to the Gerstner model:

$$\left\{ \begin{array}{lcl} X & = & X_0 + \sum_i a_i \frac{K_i}{|K_i|} \sin(\omega_i t - K_i.X_0) \\ z & = & z_0 + \sum_i \; a_i \; \cos(\omega_i t - K_i.X_0) \end{array} \right. \quad (1)$$

where $X_0 = (x_0, y_0)$ is the location of the particle at rest on the surface and z_0 its altitude at rest.

Note that the only information that needs to be stored in memory is the specification of the wave trains: particles are evaluated on the fly, and can be at different locations from one frame to the other. Surface displacement is thus evaluated much like a *procedural function* [Ebert et al. 1994].

4 Adaptive Scheme

Our main contribution is the adaptive scheme, which applies to both the geometry and the procedural animation process: Instead of simulating a regularly sampled, predetermined region of the ocean surface, we adapt the mesh sampling to the projected size of each surface element, in the spirit of surfels [Pfister et al. 2000; Stamminger and Drettakis 2001]. We also adapt the wave model in spectral space by locally filtering the waves according to the local geometric sampling rate. In addition to saving computational time, this reduces geometric aliasing.

[2] In this paper, we use capital letters to denote vectors and lower case letters to denote scalar parameters.

4.1 An adaptive surface mesh

The idea is to generate the mesh representing the ocean surface such that every element covers, at rest, the same area on the screen. This is done by subdividing the screen into a grid of quads, which are back-projected on the plane modeling the ocean surface at rest. The resulting mesh points provides the particle locations at which the procedural animation model of equation 1 is evaluated (see Figure 1).

Using this method, a camera motion induces a continuous shift of the mesh over the ocean surface, an adequate resolution being maintained everywhere in the computed image. We can note that this sampling strategy is made possible by the continuously moving nature of the ocean surface: using a similar approach for rendering a landscape, for example, would produce artifacts since the camera motion induces a shift in the sampling locations. This sampling artifact actually takes place, but is hidden by the waves animation.

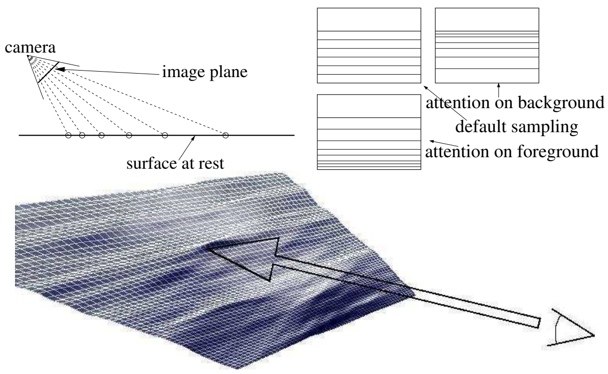


Figure 1: Dynamically adapted surface mesh, whose elements approximately have the same projected area on screen.

The use of quads allows easy factorization such as stripping. It can also help relying on hierarchical structures such as quad-tree. Lastly, focusing on a given zone of interest would be done by locally refining a given region in screen space, such as adding more bottom rows if the attention has to be focussed on the foreground (respectively more top rows for focusing on the background). See Figure 1 (top right).

The use of optimally large quads to approximate the ocean surface prevents from relying on finite difference to estimate the surface normals from vertex positions. We rather compute analytical normals at mesh vertices, obtained from the spatial derivatives of equation (1). See Figure 2.

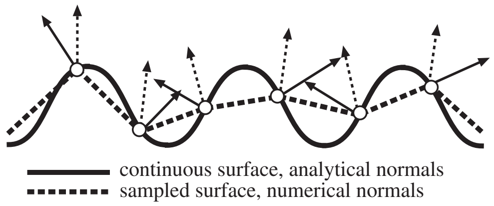


Figure 2: Due to the optimized mesh, analytical normals give much better results than normals computed using finite differences.

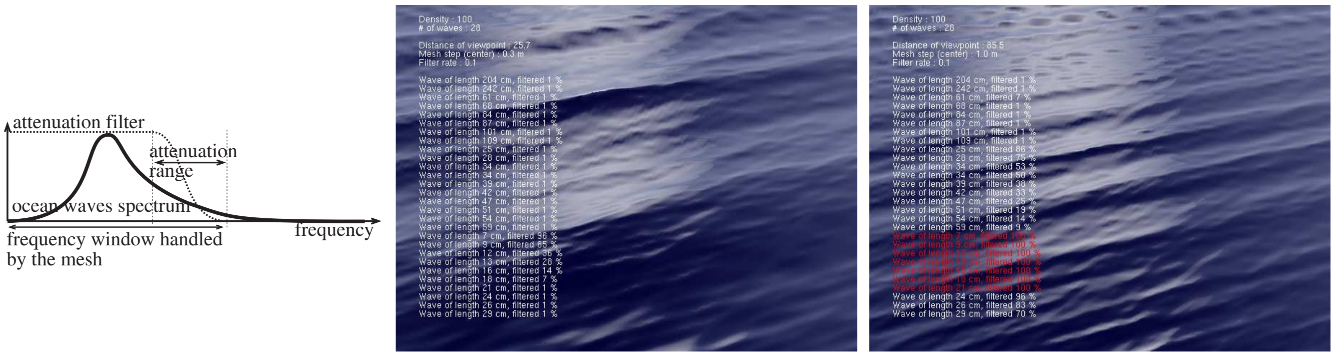


Figure 3: Filtering wave trains. *Left*: adapting the set of computed wave trains to the local mesh resolution. *Middle & right*: when increasing the distance, smaller wavelengths (in red) are no longer simulated.

4.2 Adapting the set of simulated wave trains

Since the grid defines the highest available resolution, we do not display waves which appear smaller than the grid quads. In addition to reducing aliasing, this saves much computational time, since we don't need to simulate the associated wave trains for this frame and location. For instance, a wide surface is covered by the sight near the horizon, but the resolution there is coarse. Most of the wave trains are thus pruned in this region.

Simply considering or rejecting wave trains at a given location according to their wavelength would produce popping artifacts: a wave would suddenly appear or disappear when reaching a given distance. We thus handle a transition range, where small wavelengths are filtered in order to appear or disappear progressively with the distance (see Figure 3).

Since we use many (up to 60 in our examples) wave train primitives, several of them, of different directions, use to have similar wavelengths. In practice, these could be hierarchically grouped in order to accelerate wavelength-based pruning, similarly to the hierarchy of bounding boxes used for optimizing the rendering of complex scenes.

Another useful extension would be to accelerate the evaluation of wave trains of large large wavelengths by associating a quad-tree to the screen grid, as suggested in the previous section: Gerstner formula for a given wave could then be evaluated only at the adequate level (corresponding to the scale of the wavelength), other values being interpolated. In practice, only the sum of displacements due to the larger waves would need to be stored for interpolation.

If the wave spectrum covered an infinite range, these optimizations, resulting in small wavelengths canceled on the background and large wavelengths coarsely sampled on the foreground, would result in a quasi-constant number of computed waves at any given location. In practice very small[3] and very large gravity waves do not exist. Thus, a large portion of screen only requires a small number of wave evaluations.

5 Results and Discussion

5.1 Rendering

A detailed discussion of rendering issues is out of the scope of this paper. We apply an environment map on the ocean surface, thus reflecting the sun and the sky. Moreover we modulate the amount of reflection using an approximation of the Fresnel factor. Rendering and display are thus performed using the graphics hardware.

[3]Ripples belong to a different physical phenomena, and cannot be modeled using gravity waves.

5.2 Results and timing

We implemented the filtering of waves with distance, but not the wavelengths factorization nor the quad-tree optimization suggested in Section 4.2. The bottleneck is currently in the computation of mesh-point positions and normals rather than in the mesh display, so these optimization would probably significantly increase the frame rate.

We rendered several animations with different parameter values, targeting either real-time or higher quality interactive images, as illustrated in Figure 4: 20 to 30 fps are obtained with a 50×50 screen-mesh resolution, with the horizon above the middle of the image (our test machine is a 800 MHz Pentium III PC with a GeForce2 graphics board). Good quality images can still be obtained at interactive rate (4,middle). Although it was not designed for this purpose, our model can also be used to compute high quality animations at a few seconds by frame (4,right). Other scenes are presented on Figures 6 and 7.

Compromises have to be chosen between speed and quality: using smaller quads allows for figuring thinner details with a higher cost. If the quads are large, the ocean surface might look too smooth. As for the MIP-mapping technique, the filtering thresholds can be tuned depending how much contrast – including a bit of aliasing – is preferred to smoothness. The worst case occurs on the horizon: Using large quads smoothes the waves at horizon, which prevents aliasing. On the other hand a bit of aliasing can be preferred, since real ocean horizon is glittering. In the case of quality rendering a solution based on analytical integration of sub-scales (in the spirit of [Fournier 1992]) should probably be considered.

5.3 Comparison with other approaches

Our model is dedicated to the case of deep water. As compared to the first family of models mentioned in Section 2.2, it does not handle waves refraction occurring close to the shore (the models of the second family have the same limitation). But it allows the combination of a large number of different waves, so that we can obtain a large range of surface aspects going from regular parallel wave trains to random looking surfaces. Extension to shallow water simulation could be done by replacing the circular motion of mesh vertices by elliptic motion (following Stokes and Biesel's model) and combining our waves with a precomputed shore refraction, similarly to what is done in [Gamito and Musgrave 2000].

We consider the second family of approaches (i.e, the spectral methods) as the reference for deep water waves generation, since they have successfully been used in special-effects. Providing a precise comparison with them is thus essential for showing the benefits of our approach.

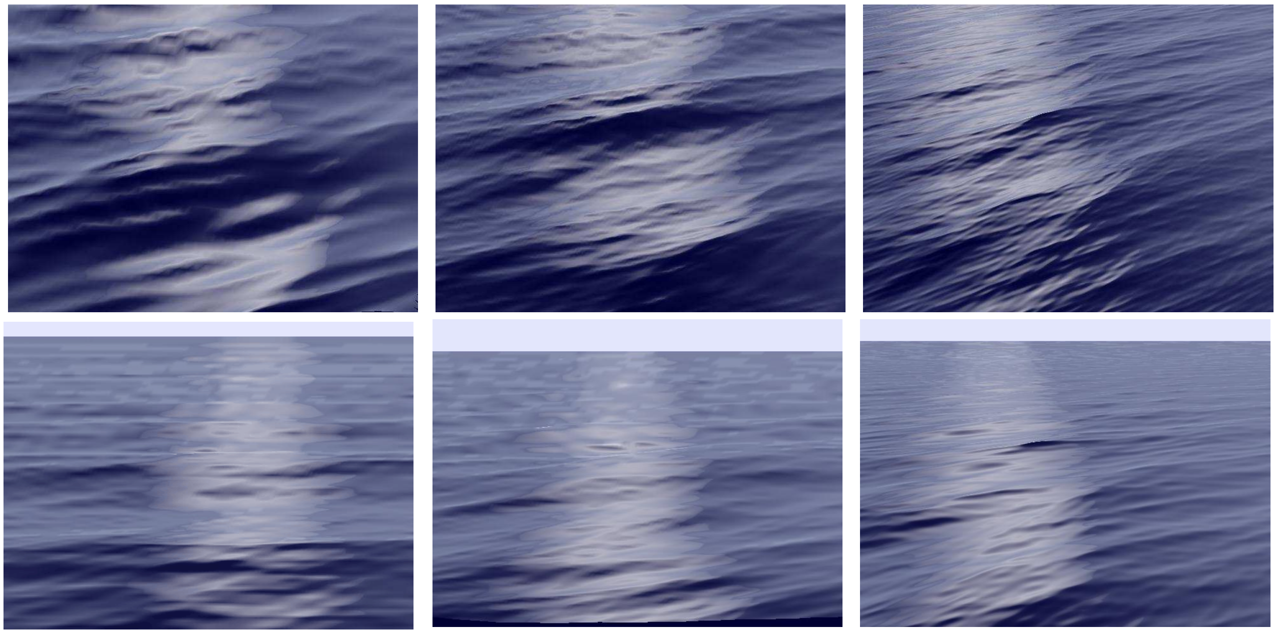


Figure 4: *Left:* Mesh resolution 50×50. 15 fps. *Middle:* Mesh resolution 100×100; 30 primitives. 3 to 5 fps. *Right:* Mesh resolution 500×500; 60 primitives. 10 seconds.

The cost of a spectral ocean simulation is mainly due to the FFT. On a $n_x \times n_y$ grid (where n_x and n_y are powers of two), its complexity is $n_x n_y \log(n_x n_y)$. The grid resolution should fit the desired amount of details in the foreground. In addition, the simulated region should be large enough to avoid obvious cyclicity (note that this region can never extend up to the horizon). Usually, the simulated rectangle is chosen large enough so that the distant edge covers the entire screen size (see Figure 5). For the ease of comparison, let us assume that the horizon projects on the top of the screen (supposedly square), and that the required mesh density on the foreground yields to n samples in the first, closest row. A simulated region of $3n \times 2n$ would then be reasonable, giving a complexity of $6n^2 \log(6n^2)$.

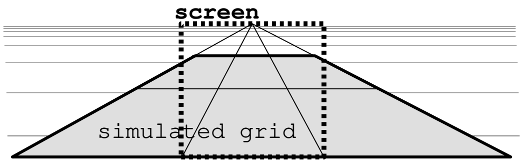


Figure 5: Simulated area for the spectral approach.

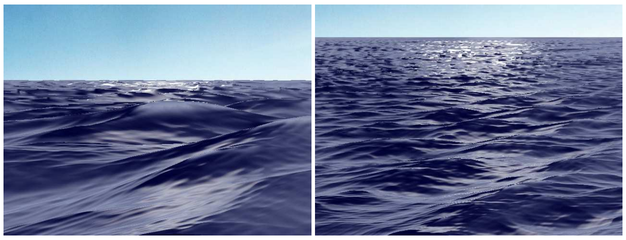


Figure 6: Stormy sea.

In the same case, our algorithm would consist in evaluating trochoids at each grid point[4], for a grid only covering the visible portion of the image. If a regular grid is used in screen space, this yields $n \times \frac{3}{4} n \times \overline{m}$ trochoid evaluations, where $\overline{m}$ is the *average* number of wave trains that our adaptive method evaluates by mesh vertex. The algorithm is thus less complex than the reference one, i.e. $O(n^2)$ instead of $O(n^2 \log n^2)$.

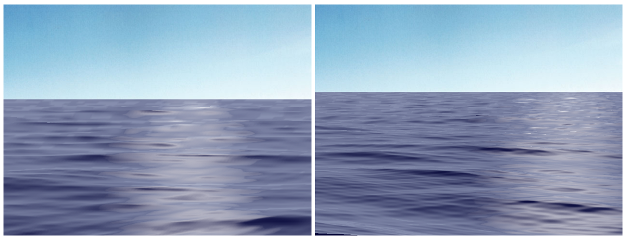


Figure 7: Calm sea. *Left*: real-time. *Right*: realistic.

Let's discuss the value of the constant $\overline{m}$ compared with the maximum number m of simulated wave trains: Since the size of grid quads, and thus the minimal considered wavelength, decreases when the distance increases, $\overline{m}$ is roughly $\frac{1}{2}m$, assuming an infinite range and a uniform repartition of the gravity waves spectrum. In practice, gravity waves do not cover the entire frequency spectrum, as explained in Section 4.2. There are thus less wave train to evaluate when approaching the horizon or the foreground, i.e. at the top and the bottom of the image. Using the suggested quad-tree optimization would, according to our estimations[5], decrease $\overline{m}$ to about $\frac{1}{12}m$ in this example. Of course, this constant factor does not change the theoretical complexity, but it illustrates that optimizations can save an order of magnitude.

In addition to increasing efficiency, our method provides more flexibility than the spectral approach: The mesh density on the screen can be tuned to increase resolution in regions of interest. Tuning the number m of wave trains provides an almost continuous control of the quality/cost ratio. Since our scheme is procedural, we can evaluate the ocean surface at any desired location. This can be useful for incorporating extra effects such as the interaction

[4] We do not explicitly mention the analytical normals evaluation, which simply doubles the amount computation for both methods.

[5] We estimate the proportion of non-filtered wave trains by integrating over the image range.

with a floating object or to ray-trace images. Other wave models can be easily incorporated into the model (e.g. ship waves, ripples, obstacles; see Figure 8) as long as they can be expressed in a procedural manner, while the spectral approaches can only deal with homogeneous phenomena on the surface. Moreover, we can simulate arbitrary ocean surfaces without showing cyclicity[6]: the ratios between wavelengths just have to be irrational numbers. Lastly, our method works at no extra cost whatever the *camera motion*, thus enabling the user to interactively fly over an unbounded ocean. This is illustrated by one of the animation sequences submitted with the paper.

Figure 8: Adding other kinds of wave is easy: here, ship waves.

6 Conclusion and Future Work

We have proposed an adaptive scheme for the interactive animation and display of ocean surfaces: relying on a procedural approach, it adapts both the geometric sampling of the surface and the set of simulated wave trains according to the distance. All computations are exclusively concentrated onto the visible part of the ocean, which yields real-time performance with a relatively good image quality, even in our non-optimized implementation.

One of the benefits of our approach is its flexibility: the use of trochoids enables to model a wide range of ocean surfaces, from calm to stormy seas. Since the displacement of a sample point is computed as a sum of wave contributions, adding extra effect such as ship waves is easy. The camera position can be arbitrarily chosen without changing the amount of computation nor the image quality (no cyclicity will appear). As a consequence, the user can interactively fly over an unbounded ocean surface, which makes the method promising for video game applications. Lastly, the quality/cost ratio is tunable, so higher-quality images can be computed using the same model.

Concerning future work, we plan to implement the optimizations suggested in Section 4.2, to include other kinds of waves such as ripples or ship waves, and to study how waves reflection and refraction could be introduced. We also plan to simulate the glittering of the ocean waves near the horizon due to the multiplicity of normals. This could be done by implementing a specific shader in the spirit of [Fournier 1992].

Acknowledgments

We wish to thank George Drettakis for rereading an early version of the paper.

[6] If necessary we could also achieve cyclicity by choosing rational ratios between wavelengths, thus enabling the use of a simulated sea region as a tile.

References

ARETÉ IMAGE SOFTWARE. http://www.areteis.com.

CRAWFORD, JR, F. 1977. *Waves*. McGraw-Hill.

DEBUNNE, G., DESBRUN, M., CANI, M.-P., AND BARR, A. 2001. Dynamic real-time deformations unsing space and time adaptive sampling. In *SIGGRAPH'01 Conference Proceedings*, Addison Wesley, Annual Conference Series, ACM SIGGRAPH. Los Angeles, CA.

EBERT, D., MUSGRAVE, K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 1994. *Texturing and Modeling: A Procedural Approach*. Academic Press, Oct.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. *Proceedings of SIGGRAPH 2001* (August), 23–30.

FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. In *Graphics Interface '96*, W. A. Davis and R. Bartels, Eds., 204–212.

FOURNIER, A., AND REEVES, W. T. 1986. A simple model of ocean waves. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, D. C. Evans and R. J. Athay, Eds., vol. 20, 75–84.

FOURNIER, A. 1992. Normal distribution functions and multiple surfaces. In *Graphics Interface '92 Workshop on Local Illumination*, 45–52.

GAMITO, M., AND MUSGRAVE, K. 2000. An accurate model of wave refraction over shallow water. In *Eurographics Workshop on Computer Animation and Simulation*, 155–171.

GONZATO, J.-C., AND SAËC, B. L. 2000. On modelling and rendering ocean scenes. *The Journal of Visualization and Computer Animation 11*, 1, 27–37.

HASSELMANN, D. E., M.DUNCKEL, AND EWING, J. A. 1980. Directional wave spectra observed during jonswap 1973. *J. Phys. Oceanogr. 10* (August), 1264–1280.

KASS, M., AND MILLER, G. 1990. Rapid, stable fluid dynamics for computer graphics. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, F. Baskett, Ed., vol. 24, 49–57.

KINSMAN, B. 1984. *Wind Waves, Their Generation and Propagation on the Ocean Surface*. Dover Publication.

LIGHTHILL, J. 1978. *Waves in fluids*. Cambridge University Press.

MASTIN, G. A., WATTERBERG, P. A., AND MAREDA, J. F. 1987. Fourier synthesis of ocean scenes. *IEEE Computer Graphics and Applications 7*, 3 (Mar.), 16–23.

NEYRET, F., AND PRAIZELIN, N. 2001. Phenomenological simulation of brooks. In *Eurographics Workshop on Computer Animation and Simulation*, Springer, Eurographics, 53–64.

PEACHEY, D. R. 1986. Modeling waves and surf. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, vol. 20, 65–74.

PERLIN, K. 1985. An image synthesizer. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, B. A. Barsky, Ed., vol. 19(3), 287–296.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: Surface elements as rendering primitives. *Proceedings of SIGGRAPH 2000* (July), 335–342.

STAMMINGER, M., AND DRETTAKIS, G. 2001. Interactive sampling and rendering for complex and procedural geometry. In *Rendering Techniques 2001 (Proceedings of the Eurographics Workshop on Rendering 01)*, Springer Verlag, K. Myskowski and S. Gortler, Eds., 12th Eurographics workshop on Rendering, Eurographics, 151–162.

STOLLNITZ, E., DEROSE, T., AND SALESIN, D. *Wavelets for Computer Graphics*. Morgan Kaufmann, San Francisco, California.

TESSENDORF, J. 1999. Simulating ocean water. In *Siggraph Course Notes*, Addison-Wesley.

THON, S., AND GHAZANFARPOUR, D. 2001. A semi-physical model of running waters. In *Eurographics UK*.

THON, S., DISCHLER, J.-M., AND GHAZANFARPOUR, D. 2000. Ocean waves synthesis using a spectrum-based turbulence function. In *Computer Graphics International Proceeding*.

TS'O, P., AND BARSKY, B. 1987. Modeling and rendering waves: Wave-tracing using beta-splines and reflective and refractive texture mapping. *ACM Transactions on Graphics 6*, 3 (July), 191–214.

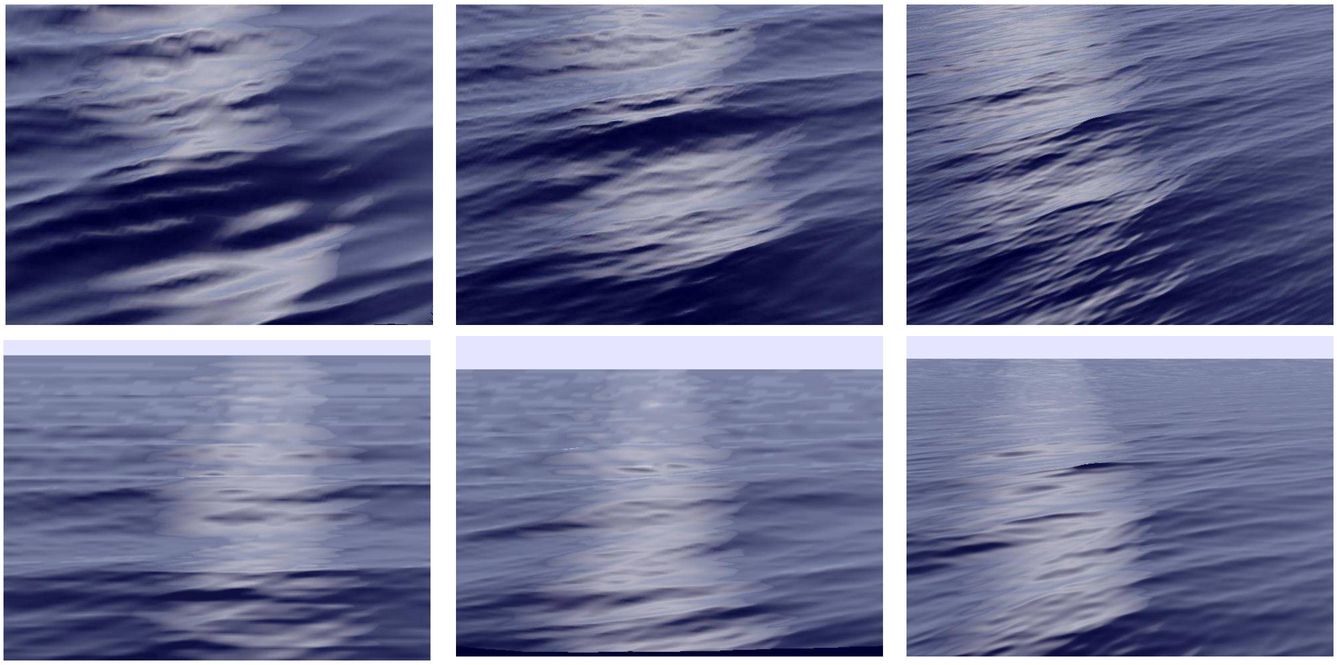


Figure 4: *Left:* Mesh resolution 50×50. 15 fps. *Middle:* Mesh resolution 100×100; 30 primitives. 3 to 5 fps. *Right:* Mesh resolution 500×500; 60 primitives. 10 seconds.

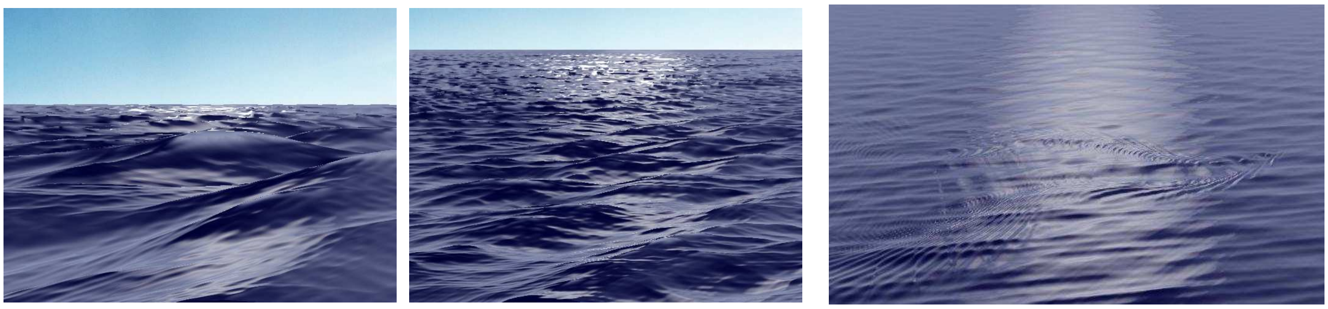


Figure 6 and 8: *Left & middle:* Stormy sea. *Right:* Adding other kinds of wave is easy: here, ship waves.

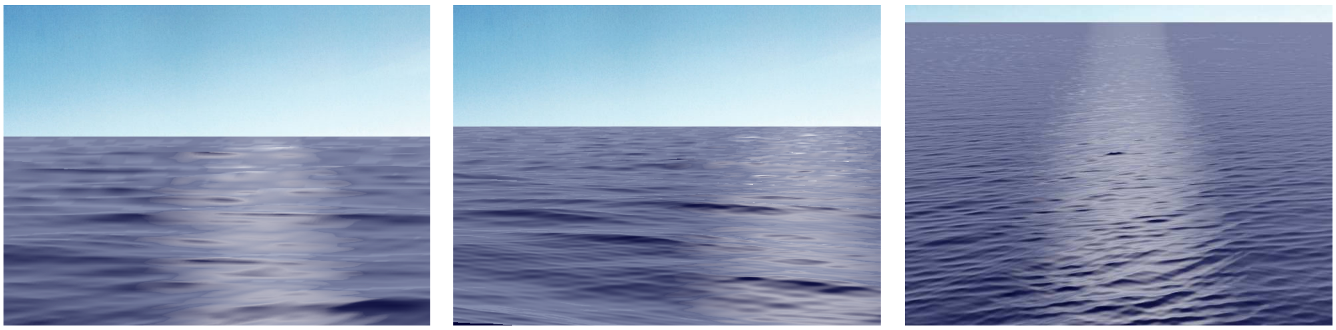


Figure 7: Calm sea. *Left*: real-time. *Middle & right*: realistic.