Simulating Ocean Water

Jerry Tessendorf

1 Introduction and Goals

These notes are intended to give computer graphics programmers and artists an introduction to methods of simulating, animating, and rendering ocean water environments. CG water has become a common tool in visual effects work at all levels of computer graphics, from print media to feature films. Several commercial products are available for nearly any computer platform and work environment. A few visual effects companies continue to extend and improve these tools, seeking to generate higher quality surface geometry, complex interatctions, and more compelling imagery. In order for an artist to exploit these tools to maximum benefit, it is important that he or she become familiar with concepts, terminology, a little oceanography, and the present state of the art.

As demonstrated by the pioneering efforts in the films *Waterworld* and *Titanic*, as well as several other films made since about 1995, images of cg water can be generated with a high degree of realism. However, this level of realism has been mostly limited to relatively calm, nice ocean conditions. Conditions with large amounts of spray, breaking waves, foam, splashing, and wakes are improving and approaching the same realistic look.

Three general approaches are currently popular in computer graphics for simulating fluid motion, and water surfaces in particular. The three methods are all related in some way to the basic Navier-Stokes equations at the heart of many applications. Computational Fluid Dynamics (CFD) is one of the methods which has recieved a great amount of attention lately. While many versions of this technology have been in existence for some time, recent papers by Stam [1], Fedkiw and Foster [2], and many others have demonstrated that the numerical computation discipline and the computer graphics discipline have connected well enough to produce useful, interesting, and sometimes beautiful results. The primary drawback of CFD methods is that the computations are performed on data structured in a 2D or 3D grid, called an Eulerian framework. This gridded architecture limits the combined extent and flow detail that can be computed. At present for example, it is practical to simulate with CFD waves breaking as they approach a shallow beach. However, that simulation is not able to simulate a bay-sized region of ocean while also simulating the breaking down to the detail of spray formation, simply because the number of grid points needed is prohibitively large.

A second method that is shows great promise is called Smoothed Particle Hydrodynamics (SPH). This is a completely different approach to solving the Navier-Stokes equations. SPH imagines that the volume of a fluid is composed of small overlapping regions, the center of each region carrying some amount of mass and momentum. The regions are allowed to move about within the fluid and experience forces due to pressure, strain, gravity, and others. But now the center of each region acts as a particle, and the Navier-Stokes equations are converted into equations of motion for discrete particles. This approach is called a Lagragian framework (as opposed to the Eulerian framework in CFD). The fluid volume is not bound to a grid geometry. SPH is a very useful method of simulation for situations in which there is significant splashing or explosions, and has even been used to simulated cracking in solids [3]. Using implicit methods to construct a surface for the fluid, standard computer graphics applications such as pouring water have been achieved [4].

Finally, the third method is the one that is the focus of these notes, and unlike the CFD and SPH methods, this one is focused on the more narrow goal of simulating the motion of the surface of a body of water. Surface simulations are commonly generated from the CFD and SPH methods, but the surface is generated by algorithms that are added to those fluid simulations, for example by tracking a type of implicit surface called a level set. In choosing to focus on the surface structure and motion, we eliminate much of the computation and resolution limits in the CFD and SPH methods. Of course when we eliminate those computations we have lost certain types of realistic motion, most notably the breakup of the surface with strong changes in topology. In place of that computation we substitute a mixture of knowledge of oceanographic phenomenology and computational flexibility to achieve realistic types of surfaces that cannot practically be achieved by the others. In that sense, this phemonological approach should be considered complementary – not competitive – to the CFD and SPH methods, since all three work best in different regimes of fluid motion.

Broadly, the reader should come away from this material with

1. an understanding of the important physical concepts for ocean surface propagation, most notably the concept of dispersion and types of dispersion relationships.

2. an understanding of some algorithms that generate/animate water surface height fields suitable for modeling waves as big as storm surges and as small as tiny capillaries;

3. an understanding of the basic optical processes of reflection and refraction from a water surface;

4. an introduction to the color filtering behavior of ocean water;

5. an introduction to complex lighting effects known as caustics and godrays, produced when sunlight passes through the rough surface into the water volume underneath; and

6. some rules of thumb for which choices make nice looking images and what are the tradeoffs of quality versus computational resources. Some example shaders are provided, and example renderings demonstrate the content of the discussion.

Before diving into it, I first want to be more concrete about what aspect of the ocean environment we cover (or not cover) in these notes. Figure 1 is a rendering of an oceanscape produced from models of water, air, and clouds. Light from the clouds is reflected from the surface. On the extreme left, sun glitter is also present. The generally bluish color of the water is due to the reflection of blue skylight, and to light coming out of the water after scattering from the volume. Although these notes do not tackle the modeling and rendering of clouds and air, there is a discussion of how skylight from the clouds and air is reflected from, or refracted through, the water surface. These notes will tell you how to make a height-field displacement-mapped surface for the ocean waves with the detail and quality shown in the figure. The notes also discuss several effects of the underwater environment and how to model/render them. The primary four effects are sunbeams (also called godrays), caustics on underwater surfaces, blurring by the scattering of light, and color filtering.

There are also many other complex and interesting aspects of the ocean environment that will not be covered. These include breaking waves, spray, foam, wakes around objects in the water, splashes from bodies that impact the surface, and global illumination of the entire ocean-atmosphere environment. There is substantial research underway on these topics, and so it is possible that future versions of this or other lecture notes will include them. I have included a brief section on advanced modifications to the basic wave height algorithm that produce choppy waves. The modification could feasibly lead to a complete description of the surface portion of breaking waves, and possibly serve to drive the spray and foam dynamics as well.

There is, of course, a substantial body of literature on ocean surface simulation and animation, both in computer graphics circles and in oceanography. One of the first descriptions of water waves in computer graphics was by Fournier and Reeves[12] , who modeled a shoreline with waves coming up on it using a water surface model called *Gerstner waves*. In that same issue, Darwin Peachey[13]

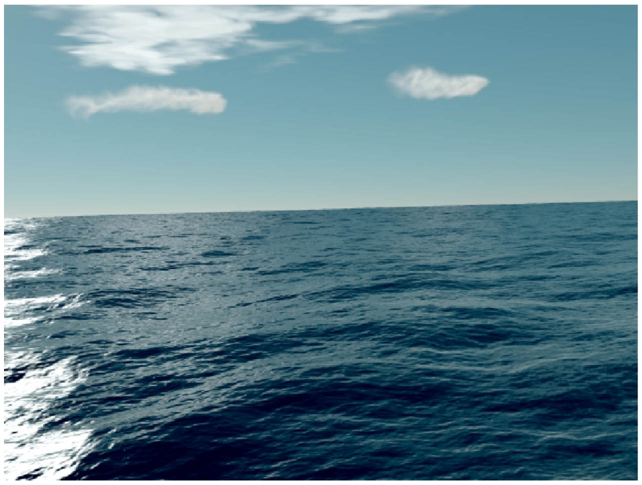

Figure 1: Rendered image of an oceanscape.

presented a variation on this approach using basis shapes other than sinusoids.

In the oceanographic literature, ocean optics became an intensive topic of research in the 1940s. S.Q. Duntley published[17] in 1963 papers containing optical data of relevance to computer graphics. Work continues today. The field of optical oceanography has grown into a mature quantitative science with subdisciplines and many different applications. One excellent review of the state of the science was written by Curtis Mobley[18].

In these lectures the approach we take to creating surface waves is close to the one outlined by Masten, Watterberg, and Mareda[11], although the technique had been in use for many years prior to their paper in the optical oceanography community. This approach synthesizes a patch of ocean waves from a Fast Fourier Transform (FFT) prescription, with user-controllable size and resolution, and which can be tiled seamlessly over a larger domain. The patch contains many octaves of sinusoidal waves that all add up at each point to produce the synthesized height. The mixture of sinusoidal amplitudes and phases however, comes from statistical, emperically-based models of the ocean. What makes these sinusoids look like waves and not just a bunch of sine waves is the large collection of sinusoids that are used, the relative amplitudes of the sinusoids, and their animation using the dispersion relation. We examine the impact of the number of sinusoids and resolution on the quality of the rendered image.

In the next section we begin the discussion of the ocean environment with a broad introduction to the global illumination problem. The radiosity equations for this environment look much like those of any other radiosity problem, although the volumetric character of some of the environmental components complicate a general implementation considerably. However, we simplify the issues by ignoring some interactions and replacing others with models generated by remote sensing data.

Practical methods are presented in section 4 for creating realizations of ocean surfaces. We present two methods, one based on a simple model of water structure and movement, and one based on summing up large numbers of sine waves with amplitudes that are related to each other based on experimental evidence. This second method carries out the sum using the technique of Fast Fourier Transformation (fft), and has been used effectively in projects for commercials, television, and motion pictures.

After the discussion of the structure and animation of the water surface, we focus on the optical properties of water relevant to the graphics problem. First, we discuss the interaction at the air-water interface: reflection and refraction. This leaves us with a simple but effective Renderman-style shader suitable for rendering water surfaces in BMRT, for example. Next, the optical characteristics of the underwater environment are explored.

Finally, please remember that these notes are a living document. Some of the discussion of the various topics is still very limited and incomplete. If you find a problem or have additional questions, please feel free to contact me at jerry@finelightvisualtechnology.com. The latest version of this course documents are hosted at http://www.finelightvisualtechnology.com

2 Radiosity of the Ocean Environment

The ocean environment, for our purposes, consists of only four components: The water surface, the air, the sun, and the water volume below the surface. In this section we trace the flow of light through the environment, both mathematically and schematically, from the light source to the camera. In general, the radiosity equations here are as coupled as any other radiosity problem. To a reasonable degree, however, the coupling can be truncated and the simplified radiosity problem has a relatively fast solution.

The light seen by a camera is dependent on the flow of light energy from the source(s) (i.e. the sun and sky) to the surface and into the camera. In addition to specular reflection of direct sunlight and skylight from the surface, some fraction of the incident light is transmitted through the surface. Ultimately, a fraction of the transmitted light is scattered by the water volume back up through the interface and into the air. Some of the light that is reflected or refracted at the surface may strike the surface a second time, producing more reflection and refraction events. Under some viewing conditions, multiple reflections and refractions can have a noticeable impact on images. For our part however, we will ignore more than one reflection or refraction from the surface at a time. This not only makes the algorithms and computation easier and faster, but also is reasonably accurate in most viewing conditions and produces visually realistic imagery.

At any point in the environment above the surface, including at the camera, the total light intensity (radiance) coming from any direction has three contributions:

$$L_{ABOVE} = rL_S + rL_A + t_U L_U \ , \qquad (1)$$

with the following definitions of the terms:

r is the Fresnel reflectivity for reflection from a spot on the surface of the ocean to the camera.

t_U is the transmission coefficient for the light L_U coming up from the ocean volume, refracted at the surface into the camera.

L_S is the amount of light coming directly from the sun, through the atmosphere, to the spot on the ocean surface where it is reflected by the surface to the camera.

L_A is the (diffuse) atmospheric skylight

L_U is the light just below the surface that is transmitted through the surface into the air.

Equation 1 has intentionally been written in a shorthand way that hides the dependences on position in space and the direction the light is traveling.

While equation 1 appears to have a relatively simple structure, the terms L_S, L_A, and L_U can in principle have complex dependencies on each other, as well on the reflectivity and transmissivity.

There is a large body of research literature investigating these dependencies in detail [19], but we will not at this point pursue these quantitative methods. But we can elaborate further on the coupling while continuing with the same shorthand notation. The direct light from the sun L_S is

$$L_S = L_{TOA} \exp\{-\tau\} \; , \tag{2}$$

where L_{TOA} is the intensity of the direct sunlight at the top of the atmosphere, and τ is the "optical thickness" of the atmosphere for the direction of the sunlight and the point on the earth. Both the diffuse atmospheric skylight L_A and the upwelling light L_U can be written as the sum of two terms:

$$\begin{aligned} L_A &= L_A^0(L_S) + L_A^1(L_U) \quad (3) \\ L_U &= L_U^0(L_S) + L_U^1(L_A) \quad (4) \end{aligned}$$

These equations reveal the potential complexity of the problem. While both L_A and L_U depend on the direct sunlight, they also depend on each other. For example, the total amount of light penetrating into the ocean comes from the direct sunlight and from the atmospheric sunlight. Some of the light coming into the ocean is scattered by particulates and molecules in the ocean, back up into the atmosphere. Some of that upwelling light in turn is scattered in the atmosphere and becomes a part of the skylight shining on the surface, and on and on. This is a classic problem in radiosity. It is not particularly special for this case, as opposed to other radiosity problems, except perhaps for the fact that the upwelling light is difficult to compute because it comes from volumetric multiple scattering.

Our approach, for the purposes of these notes, to solving this radiosity problem is straightforward: take the skylight to depend only on the light from the sun, since the upwelling contribution represents a "tertiary" dependence on the sunlight; and completely replace the equation for L_U with an empirical formula, based on scientific observations of the oceans, that depends only on the direct sunlight and a few other parameters that dictate water type and clarity.

Under the water surface, the radiosity equation has the schematic form

$$L_{BELOW} = tL_D + tL_I + L_{SS} + L_M \; , \tag{5}$$

with the meaning

t is the Fresnel transmissivity for transmission through the water surface at each point and angle on the surface.

L_D The "direct" light from the sun that penetrates into the water.

L_I The "indirect" light from the atmosphere that penetrates into the water.

L_{SS} The single-scattered light, from both the sun and the atmosphere, that is scattered once in the water volume before arriving at any point.

L_M The multiply-scattered light. This is the single-scattered light that undergoes more scattering events in the volume.

Just as for the above water case, these terms are all related to each other is relative complex ways. For example, the single scattered light depends on the direct and indirect light:

$$L_{SS} = P(tL_I) + P(tL_D) \tag{6}$$

with the quantity P being a linear functional operator of its argument, containing information about the single scattering event and the attenuation of the scattered light as it passes from the scatter point to the camera. Similarly, the multiply-scattered light is dependent on the single scattered:

$$L_M = G(tL_I) + G(tL_D) \; . \tag{7}$$

The functional schematic quantities P and G are related, since multiple scattering is just a series of single scatters. Formally, the two have an operator dependence that has the form

$$\begin{aligned} G &\sim P \otimes P \otimes \left\{ 1 + P + \frac{1}{2!} P \otimes P + \frac{1}{3!} P \otimes P \otimes P + \ldots \right\} \\ &\sim P \otimes P \otimes \exp(P) \; . \quad (8) \end{aligned}$$

At this point, the schematic representation may have outlived its usefullness because of the complex (and not here defined) meaning of the convolution-like operator $\otimes$, and because the expression for G in terms of P has created an even more schematic view in terms of an exponentiated P. So for now we will leave the schematic representation, and journey on with more concrete quantities the rest of the way through.

The formal schematic discussion put forward here does have a mathematically and physically precise counterpart. The field of study in Radiative Transfer has been applied for some time to water optics, by a large number researchers. The references cited are excellent reading for further information.

As mentioned, there is one additional radiosity scenario that can be important to ocean rendering under certain circumstances, but which we will not consider. The situation is illustrated in figure 2. Following the trail of the arrows, which track the direction light is travelling, we see that sometimes light coming to the surface (from above or below), can reflect and/or transmit through the surface more than once. The conditions which produce this behavior in significant amounts are: the wave heights must be fairly high, and the direction of viewing the waves, or the direction of the light source must be nearly grazing the surface. The higher the waves are, the less grazing the light source or camera need to be. This phenonmenon has been examined experimentally and in computer simulations. It is reasonably well understood, and we will ignore it from this point on.

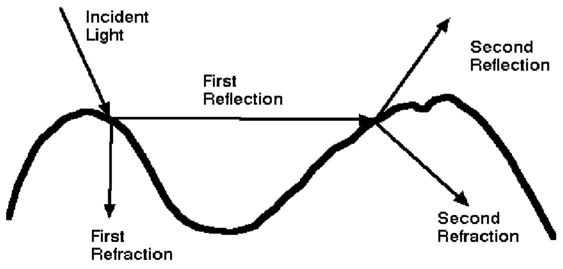

Figure 2: Illustration of multiple reflections and transmission through the air-water interface.

3 Mathematics, Physics, and Experiments on the Motion for the Surface

In this section we take a look at the mathematical problem we are trying to solve. We simplify the mathematics considerably by applying a series of approximations. How do we know these approximations are any good? There are decades of oceanographic research in which the ocean surface motion has been characterized by measurements, simulations, mathematical analysis, and experimentation. The approximations we apply in this section are not perfect, and there are many circumstances in the real world in which they break down. But they work extraordinarily well for most conditions at sea. To give you some idea of how well they work, we show some experimental work in section 4 that has been done which clearly shows these approximations at work in the real world.

3.1 Bernoulli's Equation

The starting point of the mathematical formulation of ocean surface motion is the incompressible Navier-Stokes equations. Chapter 2 of [14] provides a thorough derivation of Bernoulli's equation from Navier-Stokes. We provide here the short version, and refer you to Kinsman's or some other textbook for details.

The incompressible Navier-Stokes equations for the velocity $\mathbf{u}(\mathbf{x}, t)$ of a fluid a any position $\mathbf{x}$ at any time t are

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} &= -\nabla p + \mathbf{F} & (9) \\ \nabla \cdot \mathbf{u}(\mathbf{x}, t) &= 0 & (10) \end{aligned}$$

In these equations, $p(\mathbf{x}, t)$ is the pressure on the fluid, and $\mathbf{F}(\mathbf{x}, t)$ is the force applied to the fluid. In our application, the force is conservative, so $\mathbf{F} = -\nabla U$ for some potential energy function $U(\mathbf{x}, t)$.

For ocean surface dynamics due to conservative forces like gravity, it turns out to be worthwhile to restrict the type of motion to a class called *potential flow*. This is a situation in which the velocity has the form of a gradient:

$$\mathbf{u}(\mathbf{x}, t) = \nabla\, \phi(\mathbf{x}, t) \tag{11}$$

This restriction has the important effect of reducing the number degrees of freedom of the flow from the three components of velocity $\mathbf{u}$ to the single one of the *potential velocity* function ϕ. In fact, using this gradient form, the four Navier-Stokes equations transform into the two equations

$$\begin{aligned} \frac{\partial \phi}{\partial t} + \frac{1}{2} (\nabla \phi)^2 &= -p - U & (12) \\ \nabla^2\, \phi(\mathbf{x}, t) &= 0 & (13) \end{aligned}$$

Equation 12 is called Bernoulli's equation. As a fully nonlinear reduction of the Navier-Stokes equations, Bernoulli's equation is capable of simulating a variety of surface dynamics effects, including wave breaking in shoaling shallow water (the bottom rises from deep water up to a beach). For more detail on numerical simulations of Bernoulli's equation in 3D, see [23].

3.2 Linearization

For our purposes, we want to reduce the complexity of Bernoulli's equation even further by applying two restrictions: linearize the equations of motion, and limit evaluation of the equations to just points on the surface itself, ignoring the volume below the surface. This may seem like an extreme restriction, but when combined with some phenomenolical knowledge of the ocean, this restrictions work very well.

The first restriction is to linearize Bernoulli's equation. This is simply the task of removing the quadratic term $1/2(\nabla\phi)^2$. Eliminating this term means that we are most likely restricted to surface waves that are not extremely violent in their motion, at least in principle. So Bernoulli's equation is reduced further to

$$\frac{\partial \phi}{\partial t} = -p - U \tag{14}$$

All of the quantities ϕ, p, and U are still evaluated at 3D points $\mathbf{x}$ on the surface and in the water volume.

The second restriction is to evaluation quantities only on the water surface. To do this we have to first characterize what we mean by the surface. We will take the surface to be a dynamically changing height field, $h(\mathbf{x}_\perp, t)$, that is a function of only the horizontal position $\mathbf{x}_\perp$ and time t. For convenience, we define the mean height of the wate surface as the zero value of the height. With this definition of wave height, the gravity-induced potential energy term U is

$$U = g\, h \tag{15}$$

and g is the gravity constant, usually $9.8\ m/sec^2$ in metric units.

Restricting to just the water surface has several important consequences. One of the first consequences is for mass conservation. In the incompressible Navier-Stokes equation, mass is conserved via the mass flux equation

$$\nabla \cdot \mathbf{u}(\mathbf{x}, t) = 0 \tag{16}$$

When we chose to consider only potential flow, this mass conservation equation became

$$\nabla^2\, \phi(\mathbf{x}, t) = 0 \tag{17}$$

If we label the horizontal portion of the position vector as $\mathbf{x}_\perp$, so that $\mathbf{x} = (\mathbf{x}_\perp, y)$, and y is the coordinate pointing down into the water volume, then the mass conservation equation restricted to the surface looks in more detail like

$$\left\{ \nabla_\perp^2 \;+\; \frac{\partial^2}{\partial y^2} \right\}\, \phi(\mathbf{x}_\perp, t) = 0 \tag{18}$$

Now, when you look at this equation and see that ϕ now depends only on the $\mathbf{x}_\perp$ on the surface, you might be tempted to throw out the $\partial^2/\partial y^2$ part of the equation, because there does not appear to be a dependence. That would produce useless results. Instead, what works better in this odd world of partial differential equations is to allow ϕ to be an arbitrary function (at least with respect to this mass conservation equation) and to define the y-derivative operator to be

$$\frac{\partial}{\partial y} = \pm \sqrt{-\nabla_\perp^2} \tag{19}$$

so that the operator ∇^2 is zero. We will use this approach for any quantity evaluated on the water surface whenever we need a vertical derivative. Of course, this introduces an unusual operator that contains a square root function.

Another consequence of restricting ourselves to just the surface is that the pressure remains essentially constant, and we can choose to have that constant be 0. With this and the rest of the restrictions, Bernoulli's equation has been linearized to

$$\frac{\partial \phi(\mathbf{x}_\perp, t)}{\partial t} = -g\, h(\mathbf{x}_\perp, t) \tag{20}$$

There is one final equation that must be rewritten for this situation. Recall that the velocity potential ϕ is used to compute the 3D fluid velocity as a gradient, $\mathbf{u} = \nabla\phi$. The vertical component of the velocity must now use equation 19. In addition, the vertical velocity of the fluid is the same at the speed of the surface height. Combining these we get

$$\frac{\partial h(\mathbf{x}_\perp, t)}{\partial t} = \sqrt{-\nabla_\perp^2}\, \phi(\mathbf{x}_\perp, t) \tag{21}$$

These last two equations, 20 and 21, are the final equations of motion that are needed to solve for the surface motion. They can also be converted into a single equation. For example, if we take a derivative with respect to time of equation 21, and use 20 to substitute for the time derivative of the velocity potential, we get the single equation for the evolution of the surface height.

$$\frac{\partial^2 h(\mathbf{x}_\perp, t)}{\partial t^2} = -g\sqrt{-\nabla_\perp^2}\; h(\mathbf{x}_\perp, t) \qquad (22)$$

This still involves the unusual operator $\sqrt{-\nabla_\perp^2}$. However, taking two more time derivatives converts it to a more normal two-dimensional Laplacian for the equation

$$\frac{\partial^4 h(\mathbf{x}_\perp, t)}{\partial t^4} = g^2\nabla_\perp^2\; h(\mathbf{x}_\perp, t) \qquad (23)$$

This form is frequently the starting point for building mathematical solutions to the surface wave equation.

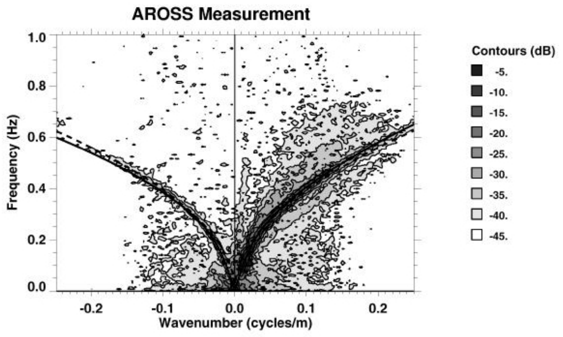

Figure 3: A slice through a 3D PSD showing that the observed wave energy follows the deep water dispersion relation very well.

3.3 Dispersion

It turns out that the equations we have built for surface height – whether in the form of equations 20 and 21, or equation 22, or equation23 – reduce to one primary lesson about surface wave propagation. This lesson is embodied in a simple mathematical relationship called the *Dispersion Relation*, which is the focus of this section. Our goal here is to obtain that simple expression from the mathematics above, understand some of its meaning, and demonstrate that, even though it appears unrealistically simplistic, the Dispersion Relation is in fact present in natural ocean waves and can be measured experimentally.

Lets just use the version of the surface evolution equation in equation 23 for convenience. The other two versions could be used and arrive at the same answer using a slightly different set of manipulations. Note that the equation of motion for the surface height is linear in the surface height. So as with any linear differential equation, the general solution of the equation is obtained by adding up any number of specific solutions. So lets find a specific solution. It turns out that all specific solutions have the form

$$h(\mathbf{x}_\perp, t) = h_0 \exp\left\{i\mathbf{k}\cdot\mathbf{x}_\perp - i\omega t\right\} \qquad (24)$$

The 2D vector $\mathbf{k}$ and the numbers ω and h_0 are generic parameters at this point. If we use this form of a solution in equation 23, it turns into an algebraic equation like this:

$$h_0\;\left\{\omega^4 - g^2k^2\right\} = 0 \qquad (25)$$

(and k is the magnitude of the vector $\mathbf{k}$). For this solution, there are only two possibilities:

1. $h_0 = 0$. Then the surface height is flat, and the solution is not very interesting.
2. $\omega = \pm\sqrt{gk}$ and h_0 can be anything. This is the interesting solution.

What we have found here is that the entire Navier-Stokes fluid dynamics problem, reduced to an evolution equation for the water surface and approximated to something that can be solved relatively easily, amounts to a single equation imposing the constraint that the temporal frequency ω of surface height movements is connected to the spatial extent of a the propagating wave $k = |\mathbf{k}|$. This relationship, $\omega = \pm\sqrt{gk}$ is the *Dispersion Relation* mentioned earlier.

Note in particular that there is no constraint placed on the amplitude h_0. But if the Navier-Stokes equation does not have anything to say about the amplitude, how do we give it a value? One way is by imposing initial conditions on the height and on its vertical speed. For ocean surface simulation in the next section, we will use an alternate method, a statistical procedure to generate random realizations of the amplitude, guided by measurements of the variance properties of wave height on the open ocean.

So the question remaining is just how reasonable is the dispersion relation for modelling realistic ocean surface waves? This is where lots of experimental research can come into play. Although there have been many decades of research on ocean wave properties using devices placed in the water to directly measure the wave motion at a point, here we look at some relatively new research that involves measuring wave properties remotely with a camera in a plane.

The AROSS [24] is a panchromatic camera mounted in a special hosing on the nose of a small airplane. Attached to the camera is navigation and GPS instrumentation which allow the camera position, viewing direction, and orientation to be measured for each frame. After the plane flys a circular orbit around a spot over the ocean, this data can be used to remap images of the ocean into a common reference frame, so that the motion of the aircraft has been removed (except for lighting variations). This remapping allows the researchers to use many frames of ocean imagery, typically 1-2 minutes worth, in some data processing to look for the dispersion relation.

The data processing that AROSS imagery is subjected to generates something called a 3D Power Spectral Density (PSD). This is obtained by taking the Fourier Transform of a time series of imagery in time, as well as Fourier Transforms in the two spatial directions of images. The output of these 3 Fourier Transforms is a quantity that is closely related to the amplitudes $h_0(\mathbf{k}, \omega)$ for each spatial and temporal frequency. These are then absolute squared and smoothed or averaged in some way so that the output is a numerical approximation of a statistical average of $|h_0(\mathbf{k}, \omega)|^2$.

But how does a 3D PSD help us decide whether the dispersion relation appears in nature? If the imagery found only dispersion constrained surface waves, then the 3D PSD should have the value 0 for all values of $\mathbf{k}$, ω that do not satisfy the dispersion relation. So mostly we would expect the 3D PSD to only have significant values in a narrow set of $\mathbf{k}$, ω values.

Figure 3 show a plot of the 3D PSD generated from AROSS images [24]. From the 3D volumetric PSD, this plot figure is a plane sliced through the volume. When sliced like this, the dispersion relation is a curve on the slice, shown as two dotted curves. The data is plotted as contours of PSD intensity, color-coded by the key on the right. PSD levels following the dispersion relation curves are much higher than in other regions. This shows that the motion of the surface waves on all scales includes a very strong dispersion relation style of motion. There are other types of motion certainly, which the PSD figure shows as intensity levels away from the dis-

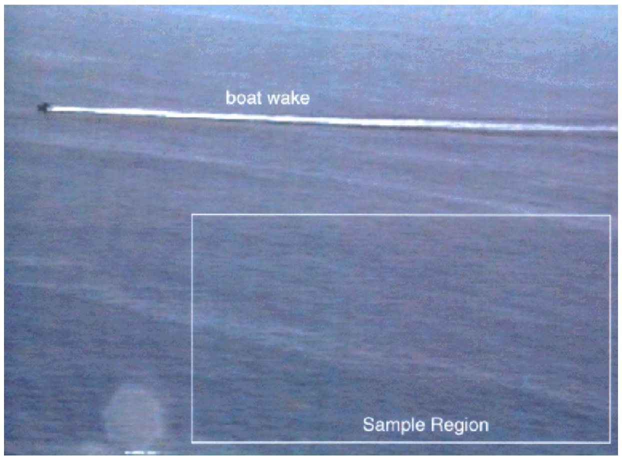

Figure 4: Site at which video data was collected in 1986, near Zuma Beach, California.

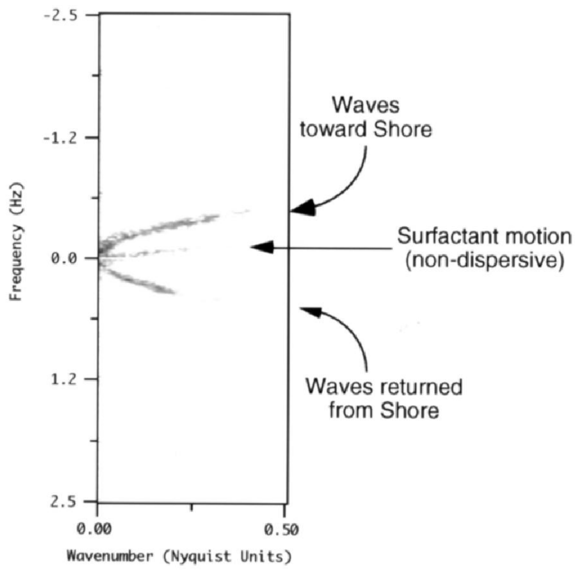

Figure 5: Slice from a 3D Power Spectral Density grayscale plot, from processed video data.

persion curves. But the dispersion motion is the strongest feature of this data.

Relatively simple experiments can be done by anyone with access to a video camera and a hilltop overlook of an ocean. For example, figure 4 is a frame from a video segment showing water coming into the beach near Zuma Beach, California. The video camera was located on hill overlooking the beach, in 1986. In 1993, the region of video frames indicated in the figure was digitized, to produce a time series of frames containing just water surface.

Figure 5 shows the actual 3D PSD from the image data. There are two clear branches along the dispersion relationship we have discussed, with no apparent modification by shallow water affects. There is also a third branch that is approximately a straight line lying between the first two. Examination of the video shows that this branch comes from a surfactant layer floating on the water in part of the video frame, and moving with a constant speed. Excluding the surface layer, this data clearly demonstrates the validity of the dispersion relationship, and demonstrates the usefulness of the linearized model of surface waves.

4 Practical Ocean Wave Algorithms

In this section we focus on algorithms and practical steps to building height fields for ocean waves. Although we will be occupied mostly by a method based on Fast Fourier Transforms (FFTs), we begin by introducing a simpler description called Gerstner Waves. This is a good starting point for several reasons: the mathematics is relatively light compared to FFTs, several important oceanographic concepts can be introduced, and they give us a chance to discuss wave animation. After this discussion of Gerstner waves, we go after the more complex FFT method, which produces wave height fields that are more realistic. These waves, called "linear waves" or "gravity waves" are a fairly realistic representation of typical waves on the ocean when the weather is not too stormy. Linear waves are certainly not the whole story, and so we discuss also some methods by which oceanographers expand the description to "nonlinear waves", waves passing over a shallow bottom, and very tiny waves about one millimeter across called capillary waves.

In the course of this discussion, we will see how quantities like windspeed, surface tension, and gravitational acceleration come into the practical implementation of the algorithms.

4.1 Gerstner Waves

Gerstner waves were first found as an approximate solution to the fluid dynamic equations almost 200 years ago. There first application in computer graphics seems to be the work by Fournier and Reeves in 1986 (cited previously). The physical model is to describe the surface in terms of the motion of individual points on the surface. To a good approximation, points on the surface of the water go through a circular motion as a wave passes by. If a point on the undisturbed surface is labelled $\mathbf{x}_0 = (x_0, z_0)$ and the undisturbed height is $y_0 = 0$, then as a single wave with amplitude A passes by, the point on the surface is displaced at time t to

$$\begin{aligned} \mathbf{x} &= \mathbf{x}_0 - (\mathbf{k}/k) A \sin(\mathbf{k}\cdot\mathbf{x}_0 - \omega t) && (26) \\ y &= A\cos(\mathbf{k}\cdot\mathbf{x}_0 - \omega t)\ . && (27) \end{aligned}$$

In these expressions, the vector $\mathbf{k}$, called the wavevector, is a horizontal vector that points in the direction of travel of the wave, and has magnitude k related to the length of the wave (λ) by

$$k = 2\pi/\lambda \qquad (28)$$

The frequency w is related to the wavevector, as discussed later.

Figure 6 shows two example wave profiles, each with a different value of the dimensionless amplitude kA. For values $kA < 1$, the wave is periodic and shows a steepening at the tops of the waves as kA approaches 1. For $kA > 1$, a loop forms at the tops of the wave, and the "insides of the wave surface are outside", not a particularly desirable or realistic effect.

As presented so far, Gerstner waves are rather limited because they are a single sine wave horizontally and vertically. However, this can be generalized to a more complex profile by summing a set of sine waves. One picks a set of wavevectors $\mathbf{k}_i$, amplitudes A_i, frequencies ω_i, and phases ϕ_i, for $i = 1, \ldots, N$, to get the expressions

$$\mathbf{x} = \mathbf{x}_0 - \sum_{i=1}^{N} (\mathbf{k}_i/k_i) A_i \sin(\mathbf{k}_i\cdot\mathbf{x}_0 - \omega_i t + \phi_i) \qquad (29)$$

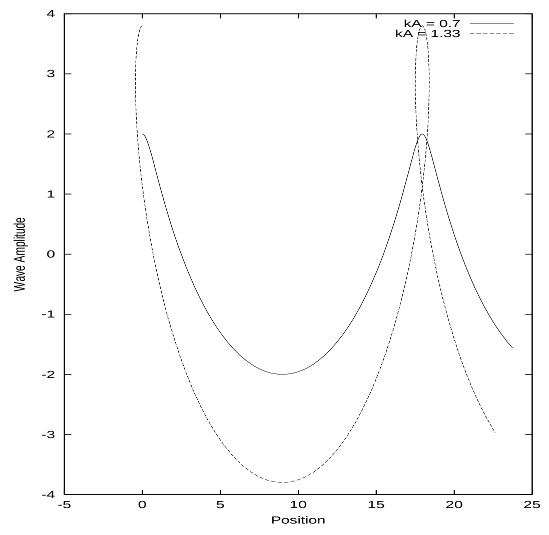

Figure 6: Profiles of two single-mode Gerstner waves, with different relative amplitudes and wavelengths.

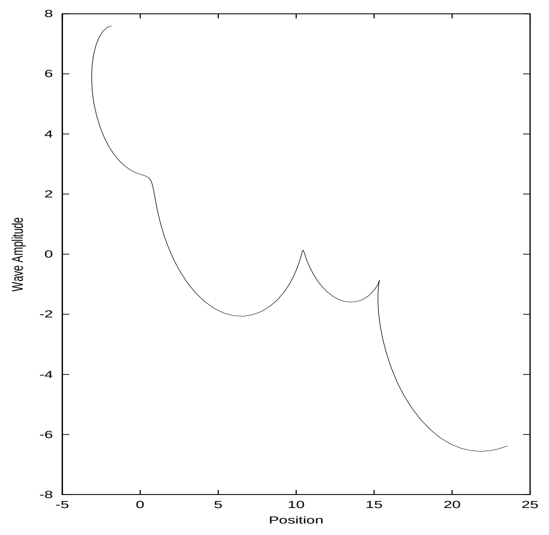

Figure 7: Profile of a 3-mode Gerstner wave.

$$y = \sum_{i=1}^{N} A_i \cos(\mathbf{k}_i \cdot \mathbf{x}_0 - \omega_i t + \phi_i) \ . \tag{30}$$

Figure 7 shows an example with three waves in the set. Interesting and complex shapes can be obtained in this way.

4.2 Animating Waves: The Dispersion Relation

The animated behavior of Gerstner waves is determined by the set of frequencies ω_i chosen for each component. For water waves, there is a well-known relationship between these frequencies and the magnitude of their corresponding wavevectors, k_i. In deep water, where the bottom may be ignored, that relationship is

$$\omega^2(k) = gk \ . \tag{31}$$

The parameter g is the gravitational constant, nominally $9.8m/sec^2$. This dispersion relationship holds for Gerstner waves, and also for the FFT-based waves introduced next.

There are several conditions in which the dispersion relationship is modified. When the bottom is relatively shallow compared to the length of the waves, the bottom has a retarding affect on the waves. For a bottom at a depth D below the mean water level, the dispersion relation is

$$\omega^2(k) = gk \tanh(kD) \tag{32}$$

Notice that if the bottom is very deep, the behavior of the $\tanh$ function reduces this dispersion relation to the previous one.

A second situation which modifies the dispersion relation is surface tension. Very small waves, with a wavelength of about 1 cm or less, have an additional term:

$$\omega^2(k) = gk(1 + k^2L^2) \ , \tag{33}$$

and the parameter L has units of length. Its magnitude is the scale for the surface tension to have effect.

Using these dispersion relationships, it is very difficult to create a sequence of frames of water surface which for a continuous loop. In order to have the sequence repeat after a certain amount of time T for example, it is necessary that all frequencies be multiples of the basic frequence

$$\omega_0 \equiv \frac{2\pi}{T} \ . \tag{34}$$

However, when the wavevectors $\mathbf{k}$ are distributed on a regular lattice, itis impossible to arrange the dispersion-generated frequencies to also be on a uniform lattce with spacing ω_0.

The solution to that is to not use the dispersion frequences, but instead a set that is close to them. For a given wavenumber k, we use the frequency

$$\bar{\omega}(k) = \left[\left[\frac{\omega(k)}{\omega_0} \right] \right] \omega_0 \ , \tag{35}$$

where $[[a]]$ means take the integer part of the value of a, and $\omega(k)$ is any dispersion relationship of interest. The frequencies $\bar{\omega}(k)$ are a *quantization* of the dispersion surface, and the animation of the water surface loops after a time T because the quantized frequencies are all integer multiples of ω_0. Figure 8 plots the original dispersion curve, along with quantized dispersion curves for two choices of the repeat time T.

4.3 Statistical Wave Models and the Fourier Transform

Oceanographic literature tends to downplay Gerstner waves as a realistic model of the ocean. Instead, statistical models are used, in combination with experimental observations. In the statistical models, the wave height is considered a random variable of horizontal position and time, $h(\mathbf{x}, t)$.

Statistical models are also based on the ability to decompose the wave height field as a sum of sine and cosine waves. The value of this decomposition is that the amplitudes of the waves have nice mathematical and statistical properties, making it simpler to build models. Computationally, the decomposition uses Fast Fourier Transforms (ffts), which are a rapid method of evaluating the sums.

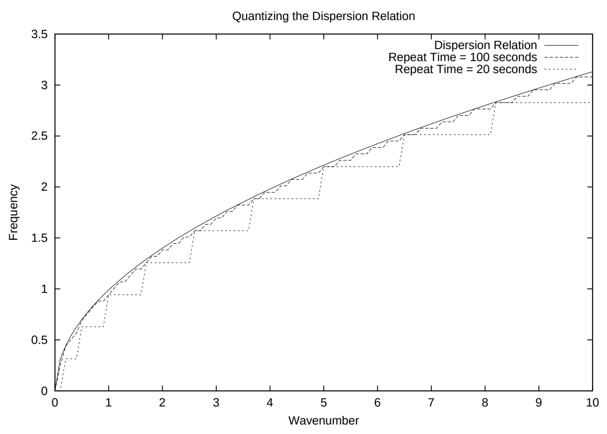

Figure 8: A comparison of the continuous dispersion curve $\omega = \sqrt{gk}$ and quantized dispersion curves, for repeat times of 20 seconds and 100 seconds. Note that for a longer repeat time, the quantized is a closer approximation to the original curve.

The fft-based representation of a wave height field expresses the wave height $h(\mathbf{x}, t)$ at the horizontal position $\mathbf{x} = (x, z)$ as the sum of sinusoids with complex, time-dependent amplitudes:

$$h(\mathbf{x}, t) = \sum_{\mathbf{k}} \tilde{h}(\mathbf{k}, t) \; \exp(i\mathbf{k} \cdot \mathbf{x}) \tag{36}$$

where t is the time and $\mathbf{k}$ is a two-dimensional vector with components $\mathbf{k} = (k_x, k_z)$, $k_x = 2\pi n/L_x$, $k_z = 2\pi m/L_z$, and n and m are integers with bounds $-N/2 \leq n < N/2$ and $-M/2 \leq m < M/2$. The fft process generates the height field at discrete points $\mathbf{x} = (nL_x/N, mL_z/M)$. The value at other points can also be obtained by switching to a *discrete* fourier transform, but under many circumstances this is unnecessary and is not applied here. The height amplitude Fourier components, $\tilde{h}(\mathbf{k}, t)$, determine the structure of the surface. The remainder of this subsection is concerned with generating random sets of amplitudes in a way that is consistent with oceanographic phenomenology.

For computer graphics purposes, the slope vector of the wave-height field is also needed in order to find the surface normal, angles of incidence, and other aspects of optical modeling as well. One way to compute the slope is though a finite difference between fft grid points, separated horizontally by some 2D vector $\Delta\mathbf{x}$. While a finite difference is efficient in terms of memory requirements, it can be a poor approximation to the slope of waves with small wavelength. An exact computation of the slope vector can be obtained by using more ffts:

$$\epsilon(\mathbf{x}, t) = \nabla h(\mathbf{x}, t) = \sum_{\mathbf{k}} i\mathbf{k} \, \tilde{h}(\mathbf{k}, t) \; \exp(i\mathbf{k} \cdot \mathbf{x}) \; . \tag{37}$$

In terms of this fft representation, the finite difference approach would replace the term $i\mathbf{k}$ with terms proportional to

$$\exp(i\mathbf{k} \cdot \Delta\mathbf{x}) - 1 \tag{38}$$

which, for small wavelength waves, does not well approximate the gradient of the wave height. Whenever possible, slope computation via the fft in equation 37 is the prefered method.

The fft representation produces waves on a patch with horizontal dimensions $L_x \times L_z$, outside of which the surface is perfectly periodic. In practical applications, patch sizes vary from 10 meters to 2 kilometers on a side, with the number of discrete sample points as high as 2048 in each direction (i.e. grids that are 2048×2048, or over 4 million waves). The patch can be tiled seamlessly as desired over an area. The consequence of such a tiled extension, however, is that an artificial periodicity in the wave field is present. As long as the patch size is large compared to the field of view, this periodicity is unnoticeable. Also, if the camera is near the surface so that the effective horizon is one or two patch lengths away, the periodicity will not be noticeable in the look-direction, but it may be apparent as repeated structures across the field of view.

Oceanographic research has demonstrated that equation 36 is a reasonable representation of naturally occurring wind-waves in the open ocean. Statistical analysis of a number of wave-buoy, photographic, and radar measurements of the ocean surface demonstrates that the wave height amplitudes $\tilde{h}(\mathbf{k}, t)$ are nearly statistically stationary, independent, gaussian fluctuations with a spatial spectrum denoted by

$$P_h(\mathbf{k}) = \left\langle \left| \tilde{h}^*(\mathbf{k}, t) \right|^2 \right\rangle \tag{39}$$

for data-estimated ensemble averages denoted by the brackets $\langle \, \rangle$.

There are several analytical semi-empirical models for the wave spectrum $P_h(\mathbf{k})$. A useful model for wind-driven waves larger than capillary waves in a fully developed sea is the *Phillips* spectrum

$$P_h(\mathbf{k}) = A \frac{\exp\left(-1/(kL)^2\right)}{k^4} |\hat{\mathbf{k}} \cdot \hat{w}|^2 \; , \tag{40}$$

where $L = V^2/g$ is the largest possible waves arising from a continuous wind of speed V, g is the gravitational constant, and $\hat{w}$ is the direction of the wind. A is a numeric constant. The cosine factor $|\hat{\mathbf{k}} \cdot \hat{w}|^2$ in the Phillips spectrum eliminates waves that move perpendicular to the wind direction. This model, while relatively simple, has poor convergence properties at high values of the wavenumber $|\mathbf{k}|$. A simple fix is to suppress waves smaller that a small length $\ell \ll L$, and modify the Phillips spectrum by the multiplicative factor

$$\exp\left(-k^2 \ell^2\right) \; . \tag{41}$$

Of course, you are free to "roll your own" spectrum to try out various effects.

4.4 Building a Random Ocean Wave Height Field

Realizations of water wave height fields are created from the principles elaborated up to this point: gaussian random numbers with spatial spectra of a prescribed form. This is most efficiently accomplished directly in the fourier domain. The fourier amplitudes of a wave height field can be produced as

$$\tilde{h}_0(\mathbf{k}) = \frac{1}{\sqrt{2}} (\xi_r + i\xi_i) \sqrt{P_h(\mathbf{k})} \; , \tag{42}$$

where ξ_r and ξ_i are ordinary independent draws from a gaussian random number generator, with mean 0 and standard deviation 1. Gaussian distributed random numbers tend to follow the experimental data on ocean waves, but of course other random number distributions could be used. For example, log-normal distributions could be used to produce height fields that are vary "intermittent", i.e. the waves are very high or nearly flat, with relatively little in between.

Given a dispersion relation $\omega(k)$, the Fourier amplitudes of the wave field realization at time t are

$$\begin{aligned} \tilde{h}(\mathbf{k}, t) &= \tilde{h}_0(\mathbf{k}) \exp\{i\omega(k)t\} \\ &+ \tilde{h}_0^*(-\mathbf{k}) \exp\{-i\omega(k)t\} \end{aligned} \tag{43}$$

This form preserves the complex conjugation property $\tilde{h}^*(\mathbf{k}, t) = \tilde{h}(-\mathbf{k}, t)$ by propagating waves "to the left" and "to the right". In addition to being simple to implement, this expression is also efficient for computing $h(\mathbf{x}, t)$, since it relies on ffts, and because the wave field at any chosen time can be computed without computing the field at any other time.

In practice, how big does the Fourier grid need to be? What range of scales is reasonable to choose? If you want to generate wave heights faster, what do you do? Lets take a look at these questions.

How big should the Fourier grid be? The values of N and M can be between 16 and 2048, in powers of two. For many situations, values in the range 128 to 512 are sufficient. For extremely detailed surfaces, 1024 and 2048 can be used. For example, the wave fields used in the motion pictures *Waterworld* and *Titanic* were 2048$\times$2048 in size, with the spacing between grid points at about 3 cm. Above a value of 2048, one should be careful because the limits of numerical accuracy for floating point calculations can become noticeable.

What range of scales is reasonable to choose? The answer to this question comes down to choosing values for L_x, L_z, M, and N. The smallest facet in either direction is $dx \equiv L_x/M$ or $dz \equiv L_z/N$. Generally, dx and dz need never go below 2 cm or so. Below this scale, the amount of wave action is small compared to the rest of the waves. Also, the physics of wave behavior below 2 cm begins to take on a very different character, involving surface tension and "nonlinear" processes. From the form of the spectrum, waves with a wavelength larger than V^2/g are suppressed. So make sure that dx and dz are smaller than V^2/g by a substantial amount (10 - 1000) or most of the interesting waves will be lost. The secret to realistic looking waves (e.g. figure 12 (a) compared to figure 12 (c)) is to have M and N as large as reasonable.

How do you generate wave height fields in the fastest time? The time consuming part of the computation is the fast fourier transform. Running on a 1+ GHz cpu, 512×512 FFTs can be generated at nearly interactive rates.

4.5 Examples: Height Fields and Renderings

We now turn to some examples of waves created using the fft approach discussed above. We will show waves in two formats: as greyscale images in which the grey level is proportional to wave height; and renderings of oceanscapes using several different rendering packages to illustrate what is possible.

In the first set of examples, the grid size is set to $M = N = 512$, with $L_x = L_z = 1000$ meters. The wind speed is a gale force at $V = 31$ meters/second, moving in the x-direction. The small-wave cutoff of $\ell = 1$ meter was also used. Figure 9 is a greyscale representation of the wave height: brighter means higher and darker means lower height. Although produced by the fft algorithms described here, figure 9 is not obviously a water height field. It may help to examine figure 10, which is a greyscale depiction of the x-component of the slope. This looks more like water waves that figure 9. What is going on?

Figures 9 and 10 demonstrate a consequence of water surface optics, discussed in the next section: the visible qualities of the surface structure tend to be strongly influenced by the slope of the waves. We will discuss this in quantitative detail, but for now we willl summarize it by saying that the reflectivity of the water is a strong function of the slope of the waves, as well as the directions of the light(s) and camera.

To illustrate a simple effect of customizing the spectrum model, figure 11 is the greyscale display of a height field identical to figure

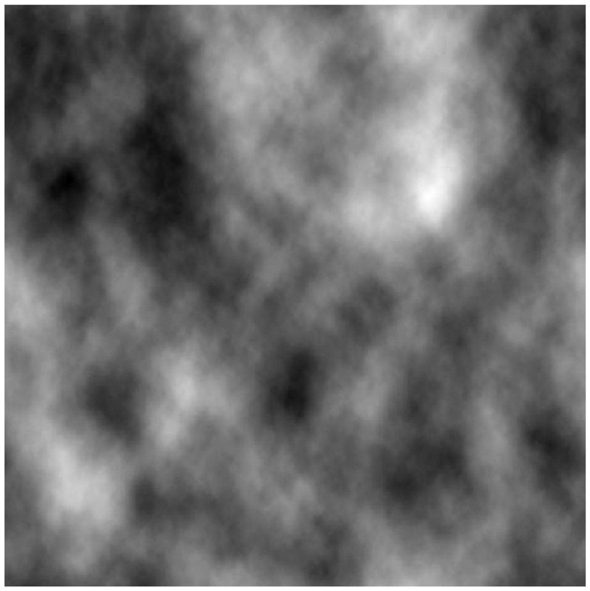
Figure 9: A surface wave height realization, displayed in greyscale.

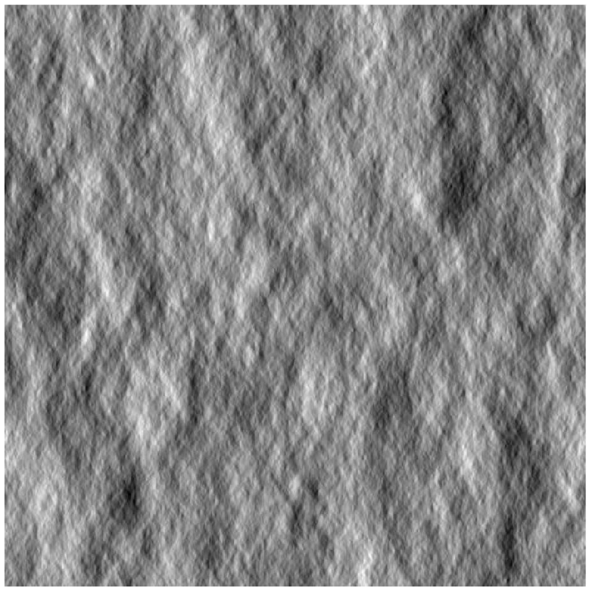
Figure 10: The x-component of the slope for the wave height realization in figure 9.

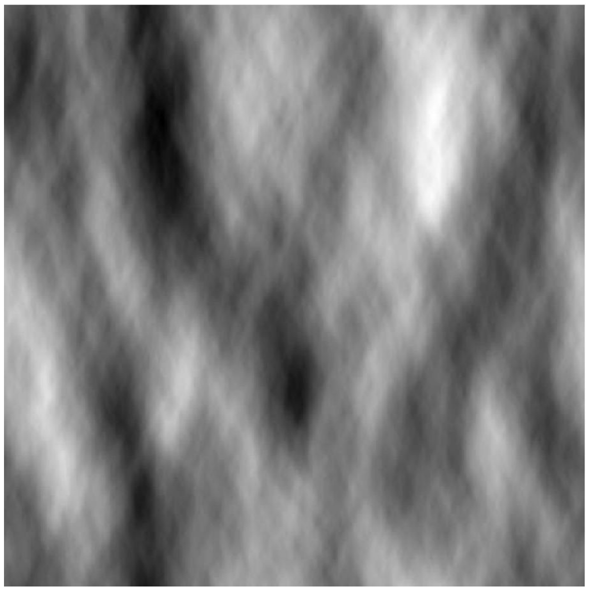

Figure 11: Wave height realization with increased directional dependence.

9, with the exception that the directional factor $|\hat{\mathbf{k}} \cdot \hat{w}|^2$ in equation 40 has been changed to $|\hat{\mathbf{k}} \cdot \hat{w}|^6$. The surface is clearly more aligned with the direction of the wind.

The next example of a height field uses a relatively simple shader in BMRT, the Renderman-compliant raytracer. The shader is shown in the next section. Figure 12 shows three renderings of water surfaces, varying the size of the grid numbers M and N and making the facet sizes dx and dz proportional to $1/M$ and $1/N$. So as we go from the top image to the bottom, the facet sizes become smaller, and we see the effect of increasing amount of detail in the renderings. Clearly, more wave detail helps to build a realistic-looking surface.

As a final example, figure 13 is an image rendered in the commercial package RenderWorld by Arete Entertainment. This rendering includes the effect of an atmosphere, and water volume scattered light. These are discussed in the next section. But clearly, wave height fields generated from random numbers using an fft prescription can produce some nice images.

4.6 Choppy Waves

We turn briefly in this section to the subject of creating choppy looking waves. The waves produced by the fft methods presented up to this point have rounded peaks and troughs that give them the appearance of fair-weather conditions. Even in fairly good weather, and particularly in a good wind or storm, the waves are sharply peaked at their tops, and flattened at the bottoms. The extent of this chopping of the wave profile depends on the environmental conditions, the wavelengths and heights of the waves. Waves that are sufficiently high (e.g. with a slope greater than about 1/6) eventually break at the top, generating a new set of physical phenonema in foam, splash, bubbles, and spray.

The starting point for this method is the fundamental fluid dynamic equations of motion for the surface. These equations are expressed in terms of two dynamical fields: the surface elevation and the velocity potential on the surface, and derive from the Navier-Stokes description of the fluid throughout the volume of the water and air, including both above and below the interface. Creamer

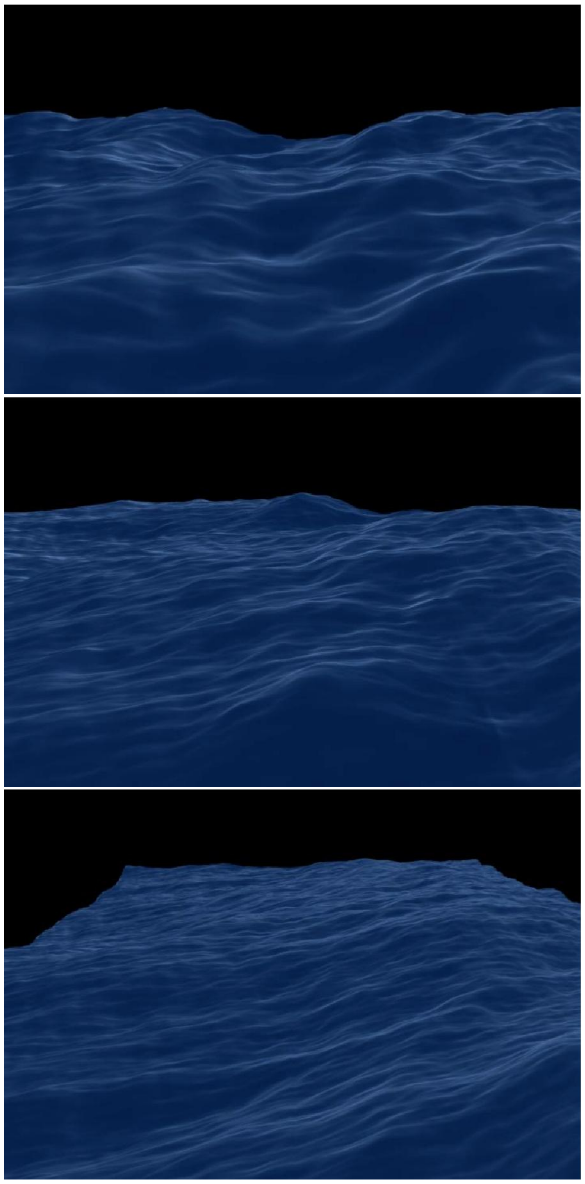

Figure 12: Rendering of waves with (top) a fairly low number of waves (facet size = 10 cm), with little detail; (middle) a reasonably good number of waves (facet size = 5 cm); (bottom) a high number of waves with the most detail (facet size = 2.5 cm).

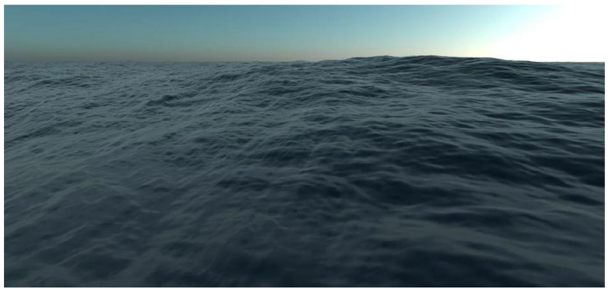

Figure 13: An image of a wave height field rendered in a commercial package with a model atmosphere and sophisticated shading.

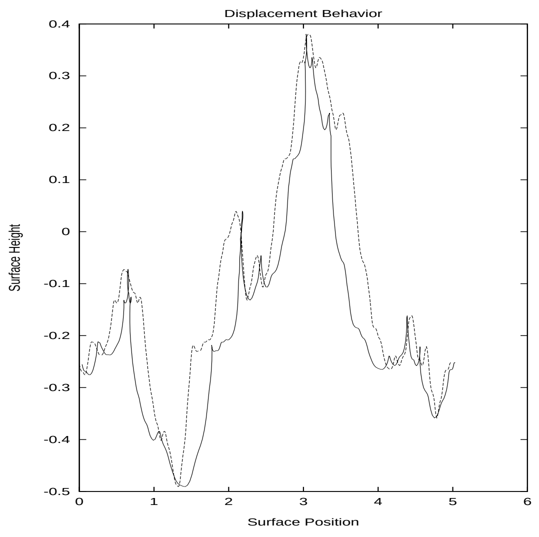

Figure 14: A comparison of a wave height profile with and without the displacement. The dashed curve is the wave height produced by the fft representation. The solid curve is the height field displaced using equation 44.

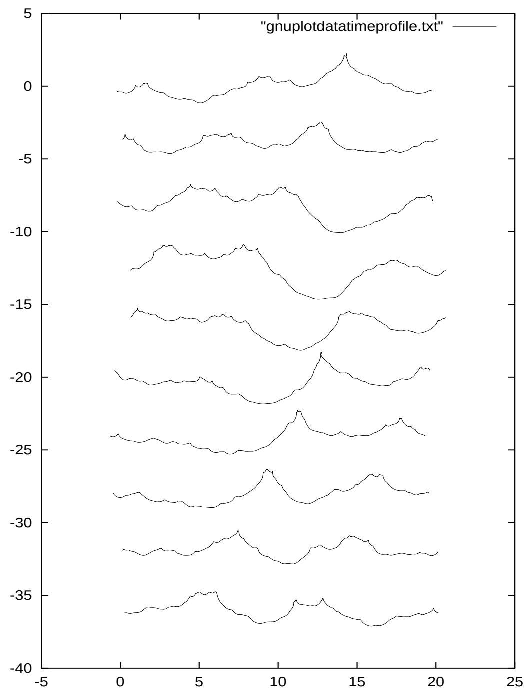

Figure 15: A time sequence of profiles of a wave surface. From top to bottom, the time between profiles is 0.5 seconds.

et al[16] set out to apply a mathematical approach called the ”Lie Transform technique” to generate a sequence of ”canonical transformations” of the elevation and velocity potential. The benefit of this complex mathematical procedure is to convert the elevation and velocity potential into new dynamical fields that have a simpler dynamics. The transformed case is in fact just the simple ocean height field we have been discussing, including evolution with the same dispersion relation we have been using in this paper. Starting from there, the inverse Lie Transform in principle converts our phenomenological solution into a dynamically more accurate one. However, the Lie Transform is difficult to manipulate in 3 dimensions, while in two dimensions exact results have been obtained. Based on those exact results in two dimensions, an extrapolation for the form of the 3D solution has been proposed: a horizontal displacement of the waves, with the displacement locally varying with the waves.

In the fft representation, the 2D displacement vector field is computed using the Fourier amplitudes of the height field, as

$$\mathbf{D}(\mathbf{x},t) = \sum_{\mathbf{k}} -i\frac{\mathbf{k}}{k}\,\tilde{h}(\mathbf{k},t)\,\exp\left(i\mathbf{k}\cdot\mathbf{x}\right) \tag{44}$$

Using this vector field, the horizontal position of a grid point of the surface is now $\mathbf{x}+\lambda\mathbf{D}(\mathbf{x},t)$, with height $h(\mathbf{x})$ as before. The parameter λ is not part of the original conjecture, but is a convenient method of scaling the importance of the displacement vector. This conjectured solution does not alter the wave heights directly, but instead warps the horizontal positions of the surface points in a way that depends on the spatial structure of the height field. The particular form of this warping however, actually sharpens peaks in the height field and broadens valleys, which is the kind of nonlinear behavior that should make the fft representation more realistic. Figure 14 shows a profile of the wave height along one direction in a simulated surface. This clearly shows that the “displacement conjecture” can dramatically alter the surface.

The displacment form of the this solution is similar to the algorithm for building Gerstner waves [12] discussed in section 4. In that case however, the displacement behavior, applied to sinusoid shapes, was the principle method of characterizing the water surface structure, and here it is a modifier to an already useful wave height representation.

Figure 15 illustrates how these choppy waves behave as they evolve. The tops of waves form a sharp cusp, which rounds out and disappears shortly afterward.

One ”problem” with this method of generating choppy waves can be seen in figure 14. Near the tops of some of the waves, the surface actally passes through itself and inverts, so that the outward normal to the surface points inward. This is because the amplitudes of the wave components can be large enough to create large displacements that overlap. This is easily defeated simply by reducing the magnitude of the scaling factor λ. For the purposes of computer graphics, this might actually be a useful effect to signal the production of spray, foam and/or breaking waves. We will not discuss here how to carry out such an extension, except to note that in order to use this region of overlap, a simple and quick test is needed for deciding that the effect is taking place. Fortunately, there is such a simple test in the form of the Jacobian of the transformation from $\mathbf{x}$ to $\mathbf{x}+\lambda\mathbf{D}(\mathbf{x},t)$. The Jacobian is a measure of the uniqueness of the transformation. When the displacement is zero, the Jacobian is 1. When there is displacement, the Jacobian has the form

$$J(\mathbf{x}) = J_{xx}J_{yy} - J_{xy}J_{yx}\ , \tag{45}$$

with individual terms

$$\begin{aligned} J_{xx}(\mathbf{x}) &= 1+\lambda\frac{\partial D_x(\mathbf{x})}{\partial x} \\ J_{yy}(\mathbf{x}) &= 1+\lambda\frac{\partial D_y(\mathbf{x})}{\partial y} \end{aligned}$$

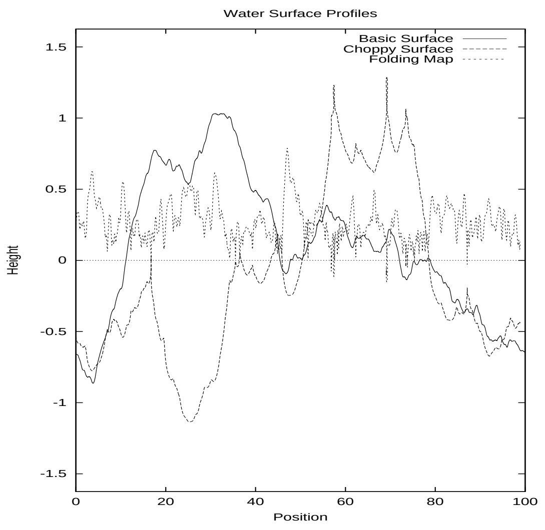

Figure 16: Wave height profile with and without the displacement. Also plotted is the Jacobian map for choppy wave profile.

$$\begin{aligned} J_{yx}(\mathbf{x}) &= \lambda \frac{\partial D_y(\mathbf{x})}{\partial x} \\ J_{xy}(\mathbf{x}) &= \lambda \frac{\partial D_x(\mathbf{x})}{\partial y} = J_{yx} \end{aligned}$$

and $\mathbf{D} = (D_x, D_y)$. The Jacobian signals the presence of the overlapping wave bacause its value is less than zero in the overlap region. For example, figure 16 plots a profile of a basic wave without displacement, the wave with displacement, and the value of J for the choppy wave (labeled ”Folding Map”). The ”folds” or overlaps in the choppy surface are clearly visible, and align with the regions in which $J < 0$. With this information, it should be relatively easy to extract the overlapping region and use it for other purposes if desired.

But there is more that can be learned from these folded waves from a closer examination of this folding criterion. The Jacobian derives from a 2×2 matrix which measures the local uniqueness of the choppy wave map $\mathbf{x} \rightarrow \mathbf{x} + \lambda\mathbf{D}$. This matrix can in general be written in terms of eigenvectors and eigenvalues as:

$$J_{ab} = J_- \hat{e}^-_a \hat{e}^-_b + J_+ \hat{e}^+_a \hat{e}^+_b \ , \ \ (a, b = x, y) \tag{46}$$

where J_- and J_+ are the two eigenvalues of the matrix, ordered so that $J_- \leq J_+$. The corresponding orthonormal eigenvectors are $\hat{e}^-$ and $\hat{e}^+$ respectively. From this expression, the Jacobian is just $J = J_- J_+$.

The criterion for folding that $J < 0$ means that $J_- < 0$ and $J_+ > 0$. So the minimum eigenvalue is the actual signal of the onset of folding. Further, the eigenvector $\hat{e}^-$ points in the horizontal direction in which the folding is taking place. So, the prescription now is to watch the minimum eigenvalue for when it becomes negative, and the alignment of the folded wave is parallel to the minimum eigenvector.

We can illustrate this phenomenon with an example. Figures 17 and 18 show two images of an ocean surface, one without choppy waves, and the other with the choppy waves strongly applied. These two surfaces are identical except for the choppy wave algorithm. Figure 19 shows the wave profiles of both surfaces along a slice through the surfaces. Finally, the profile of the choppy wave is plotted together with the value of the minimum eigenvalue in figure 20, showing the clear connection between folding and the negative value of J_-.

Incidentally, computing the eigenvalues and eigenvectors of this matrix is fast because they have analytic expressions as

$$J_\pm = \frac{1}{2}(J_{xx} + J_{yy}) \pm \frac{1}{2} \left\{ (J_{xx} - J_{yy})^2 + 4J_{xy}^2 \right\}^{1/2} \tag{47}$$

for the eigenvalues and

$$\hat{e}^\pm = \frac{(1, q_\pm)}{\sqrt{1 + q_\pm^2}} \tag{48}$$

with

$$q_\pm = \frac{J_\pm - J_{xx}}{J_{xy}} \tag{49}$$

for the eigenvectors.

5 Interactive Waves from Disturbances

The Fourier based approach to water surface evolution described in the section 4 has several limitations that make in unworkable for really interactive applications. Fundamentally, the Fourier method computes the wave height everywhere in one FFT computation. You cannot choose to compute wave height in limited areas of a surface grid without completely altering the calculation. You either get the whole surface, or you don’t compute it. This makes it very hard to customize the wave propagation problem from one location to another. So it you want to put some arbitrarily-shaped object in the water, move it around some user-constructed path, the FFT method makes it difficult to compute the wave response to the shape and movement of the object. So if you want to have an odd looking craft moving around in the water, and maybe going up and down and changing shape, the FFT method is not the easiest thing to use. Also, if you want to have a shallow bottom, with a sloping beach or underwater sea mound or some other variability in the depth of the water, the FFT method again is a challenge use.

Fortunately in the last few years an alternative scheme has emerged which allows fast construction of a water surface in response to interactions with objects in the water and/or variable bottom depth. The heart of the method is an approach to computing the propagation which does not use the FFT method at all. The fundamental mathematics of this interactive method is described in the reference [22], and is refered to as *iWave*. Here we very quickly run through the approach and show some examples in action.

At its heart the iWave method returns to the linearized equation for the wave height, 22. We can turn the time derivatives into finite time differences for a time step Δt, and get an explicit expression for the wave height:

$$\begin{aligned} h(\mathbf{x}, t + \Delta t) &= 2h(\mathbf{x}, t) - h(\mathbf{x}, t - \Delta t) \\ &- g(\Delta t)^2 \sqrt{-\nabla^2}\, h(\mathbf{x}, t) \end{aligned} \tag{50}$$

This form can be used to explicitly advance the surface wave height from one frame to the next. The hard part of course is figuring out how to calculate the last term, with the square root of the Laplacian operator.

The solution is to use convolution. The wave height is kept on a rectangular grid of points (the dimensions of which do *not* have to be powers of two), and we make use of the fact that any linear operation on a grid of data values can be converted into a convolution of some sort. The details of the numerical implementation are contained in [22].

But the real, amazing, property of iWave is that wave interaction with objects on the water surface is evaluated with a very simple

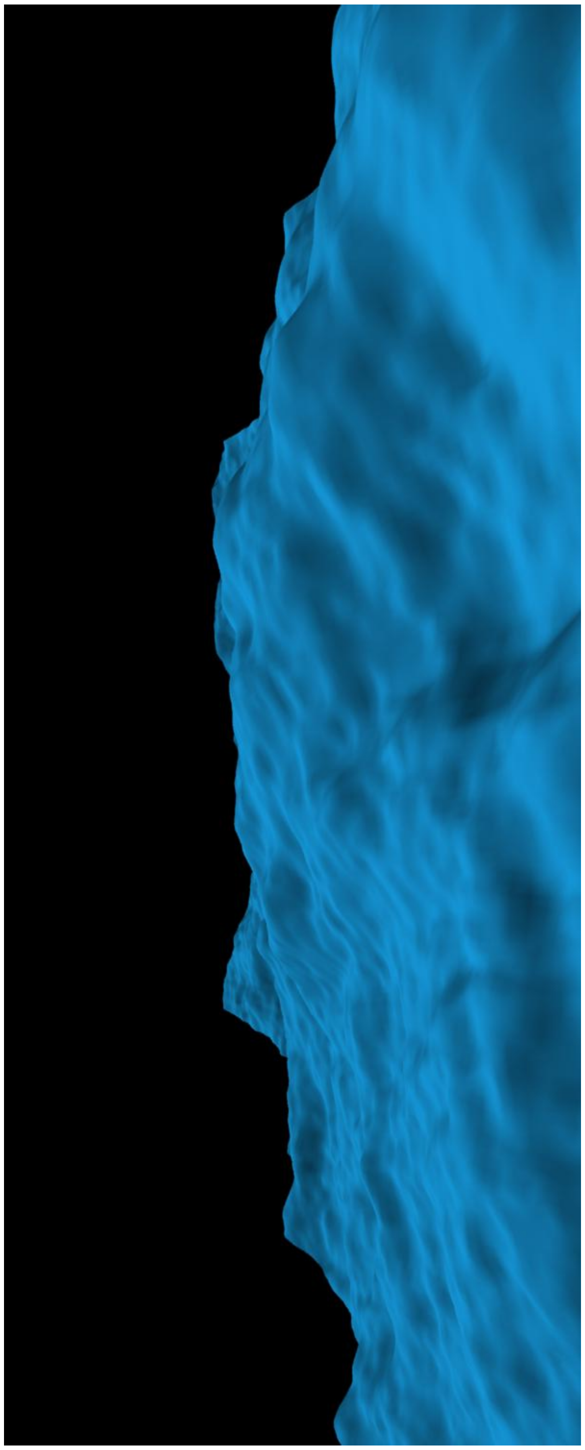
Figure 17: Simulated wave surface without the choppy algorithm applied. Rendered in BMRT with a generic plastic shader.

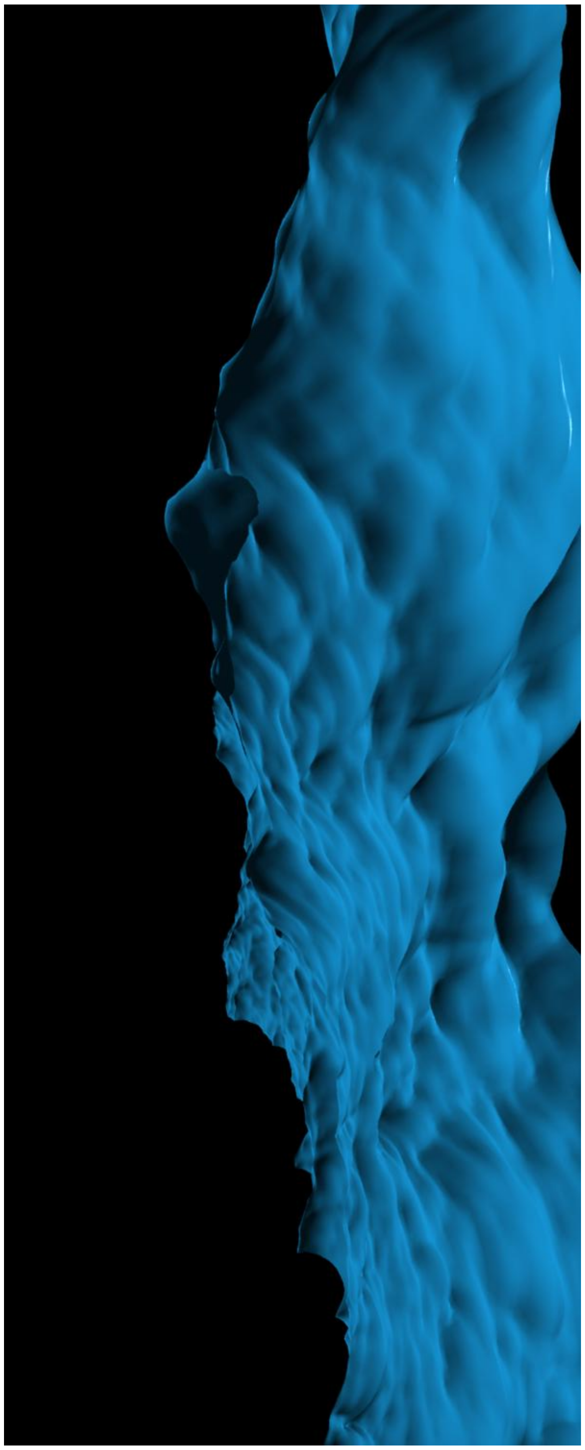
Figure 18: Same wave surface with strong chop applied. Rendered in BMRT with a generic plastic shader.

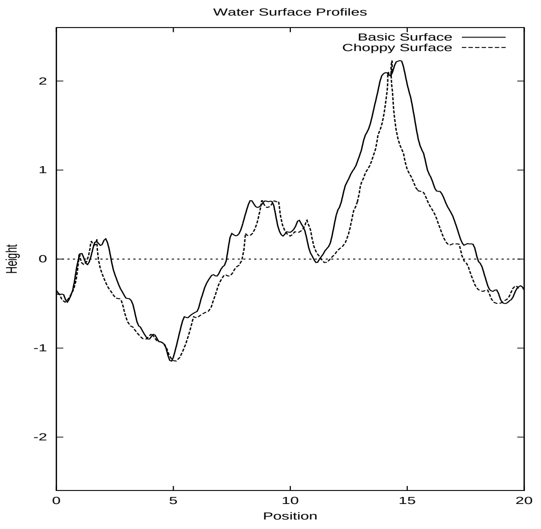

Figure 19: Profiles of the two surfaces, showing the effect of the choppy mapping.

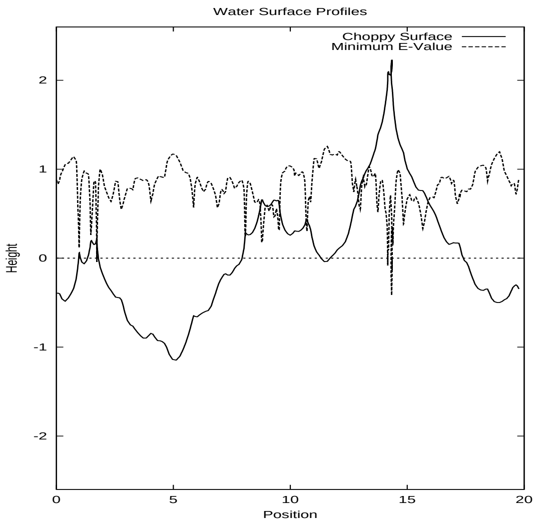

Figure 20: Plot of the choppy surface profile and the minimum eigenvalue. The locations of folds of the surface are clearly the same as where the eigenvalue is negative.

linear operation that amount to an image masking. For this purpose, an object in the water is described as a grid of mask values, zero meaning the object is present, 1 if the object is not present, and along the edges of the object values between 0 and 1 are useful as an antialiasing. Applying this mask to the wave height grid, the waves are effectively removed at grid points that the object is located at. With just this simple procedure, waves that are incident on the object reflect off of it in a realistic way. Figure 21 is a frame from a sequence, showing the wave height computed using iWave with several irregularly shaped objects.

Ordinarily, simulating waves that interact with an object on the water surface should involve a careful treatment of boundary conditions, matching water motion to the object on the boundary of the geometry, and enforcing no-slip conditions. This is a complex and time consuming computation that involves a certain amount of numerical black art. It is suprising that a simple process like masking the wave height, as described above, should not be sufficient. Yet, with the iWave procedure, correct-looking reflection/refraction waves happen in the simulation. It is not yet clear just how quantitatively accurate these interactions are. Figure 22 shows a rendered scene with a high resolution calculation of waves interacting with the hull of the ship.

5.1 Modifications for shallow water

Wave simulations based on the FFT method can simulate shallow water effects by using the dispersion relationship in equation 32. This only applies to a *flat* bottom. It would be nice to simulate wave propagation onto a beach, or pasta shallow subsurface sea mount, or over a submarine that is just below the surface. The iWave method is a great way of doing those things. Recall that the iWave method converts the mathematical operation

$$g\sqrt{-\nabla^2}\ h \tag{51}$$

into a convolution, so that effectively $g\sqrt{-\nabla^2}$ becomes a convolution kernel. A shallow bottom with depth D changes this term to (compare with the dispersion relationship)

$$g\sqrt{-\nabla^2}\ \tanh\left(\sqrt{-\nabla^2}D\right)\ \ h \tag{52}$$

This operation can also be converted into a convolution, with $g\sqrt{-\nabla^2}\ \tanh\left(\sqrt{-\nabla^2}D\right)$ becomes a convolution kernel.

How is this applied to a variable-depth bottom? The convolution kernel is a 13x13 matrix[22]. So we could build a collection of kernels over a range of depth values D. At each point on the grid, a custom kernel is constructed for the actual depth at that grid point by interpolating from the set of prebuilt values. Figure 23 is the wave height from a simulation using this technique. The bottom slopes from deep on the right hand side, to a depth of 0 on the left edge. In addition, there is a subsurface sea mount on the right.

There are three important behaviors in this simulation that occur in real shallow water propagation:

1. Waves in shallow regions have large amplitudes that in deep regions.

2. As waves approach a beach, they pile up together and have a higher spatial frequency.

3. The subsurface sea mount causes a diffraction of waves.

In addition to these capabilities, iWave can also compute other quantities, such as cuspy waves and surface velocity.

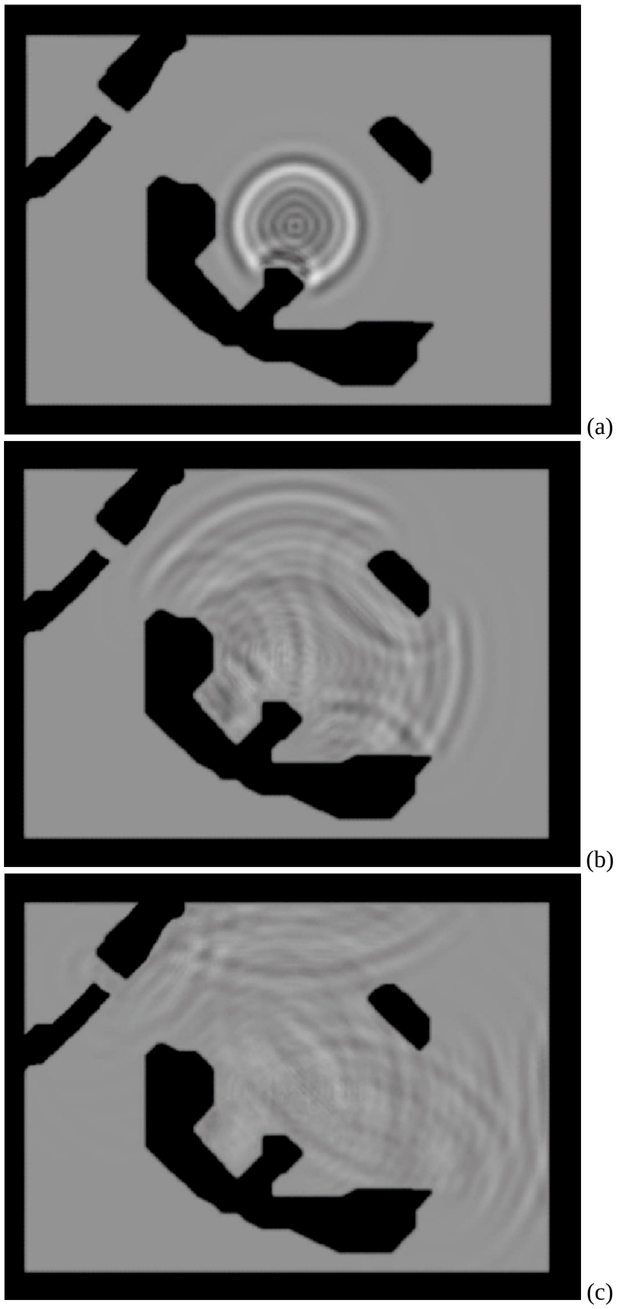

Figure 21: A sequence of frames from an iWave simulation, showing waves reflecting off of objects in the water. The objects are the black regions. In the upper left there is also diffraction taking place as waves propagate the narrow channel and emerge in the corner region.

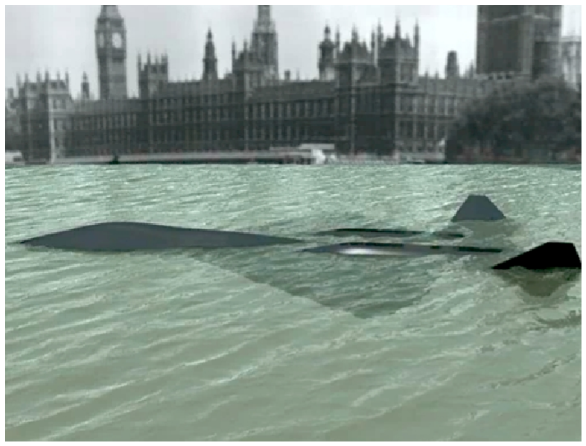

Figure 22: Frame from a simulation and rendering showing waves interacting with a ship.

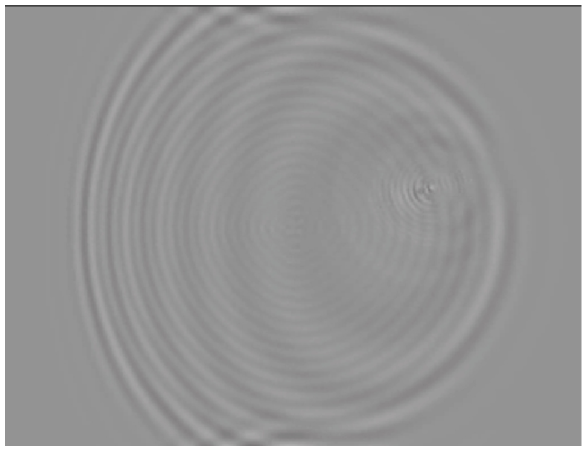

Figure 23: Frame from an iWave simulation with a variable depth shallow bottom.

6 Surface Wave Optics

The optical behavior of the ocean surface is fairly well understood, at least for the kinds of quiescent wave structure that we consider in these notes. Fundamentally, the ocean surface is a near perfect specular reflector, with well-understand relectivity and transmissity functions. In this section these properties are summarized, and combined into a simple shader for Renderman. There are circumstances when the surface does not appear to be a specular reflector. In particular, direct sunlight reflected from waves at a large distance from the camera appear to be spread out and made diffuse. This is due to the collection of waves that are smaller than the camera can resolve at large distances. The mechanism is somewhat similar to the underlying microscopic reflection mechnanisms in solid surfaces that lead to the Torrance-Sparrow model of BRDFs. Although the study of glitter patterns in the ocean was pioneered by Cox and Munk many years ago, the first models of this BRDF behavior that I am aware of were developed in the early 1980's. At the end of this section, we introduce the concepts and conditions, state the results, and ignore the in-between analysis and derivation.

Throughout these notes, and particularly in this section, we ignore one optical phenomenon completely: polarization. Polarization effects can be strong at a boundary interface like a water surface. However, since most of computer graphics under consideration ignores polarization, we will continue in that long tradition. Of course, interested readers can find literature on polarization effects at the air-water interface.

6.1 Specular Reflection and Transmission

Rays of light incident from above or below at the air-water interface are split into two components: a transmitted ray continuing through the interface at a refracted angle, and a reflected ray. The intensity of each of these two rays is diminished by reflectivity and transmissivity coefficients. Here we discussed the directions of the two outgoing rays. In the next subsection the coefficients are discussed.

6.1.1 Reflection

As is well known, in a perfect specular reflection the reflected ray and the incident ray have the same angle with respect to the surface normal. This is true for all specular reflections (ignoring roughening effects), regardless of the material. We build here a compact expression for the outgoing reflected ray. First, we need to build up some notation and geometric quantities.

The three-dimensional points on the ocean surface can be labelled by the horizontal position $\mathbf{x}$ and the waveheight $h(\mathbf{x},t)$ as

$$\mathbf{r}(\mathbf{x},t) = \mathbf{x} + \hat{\mathbf{y}} h(\mathbf{x},t) \ , \tag{53}$$

where $\hat{\mathbf{y}}$ is the unit vector pointing straight up. At the point $\mathbf{r}$, the normal to the surface is computed directly from the surface slope $\epsilon(\mathbf{x},t) \equiv \nabla h(\mathbf{x},t)$ as

$$\hat{\mathbf{n}}_S(\mathbf{x},t) = \frac{\hat{\mathbf{y}} - \epsilon(\mathbf{x},t)}{\sqrt{1+\epsilon^2(\mathbf{x},t)}} \tag{54}$$

For a ray intersecting the surface at $\mathbf{r}$ from direction $\hat{\mathbf{n}}_i$, the direction of the reflected ray can depend only on the incident direction and the surface normal. Also, as mentioned before, the angle between the surface normal and the reflected ray must be the same as the angle between incident ray and the surface normal. You can verify for yourself that the reflected direction $\hat{\mathbf{n}}_r$ is

$$\hat{\mathbf{n}}_r(\mathbf{x},t) = \hat{\mathbf{n}}_i - 2\hat{\mathbf{n}}_S(\mathbf{x},t)\left(\hat{\mathbf{n}}_S(\mathbf{x},t)\cdot\hat{\mathbf{n}}_i\right) \ . \tag{55}$$

Note that this expression is valid for incident ray directions on either side of the surface.

6.1.2 Transmission

Unfortunately, the direction of the transmitted ray is not expressed as simply as for the reflected ray. In this case we have two guiding principles: the transmitted direction is dependent only on the surface normal and incident directions, and Snell's Law relating the sines of the angles of the incident and transmitted angles to the indices of refraction of the two materials.

Suppose the incident ray is coming from one of the two media with index of refraction n_i (for air, $n = 1$, for water, $n = 4/3$ approximately), and the transmitted ray is in the medium with index of refraction n_r. For the incident ray at angle θ_i to the normal,

$$\sin\theta_i = \sqrt{1-(\hat{\mathbf{n}}_i\cdot\hat{\mathbf{n}}_S)^2} = |\hat{\mathbf{n}}_i \times \hat{\mathbf{n}}_S| \tag{56}$$

the transmitted ray will be at an angle θ_t with

$$\sin\theta_t = |\hat{\mathbf{n}}_t \times \hat{\mathbf{n}}_S| \ . \tag{57}$$

Snell's Law states that these two angles are related by

$$n_i \sin\theta_i = n_t \sin\theta_t \ . \tag{58}$$

We now have all the pieces needed to derive the direction of transmission. The direction vector can only be a linear combination of $\hat{\mathbf{n}}_i$ and $\hat{\mathbf{n}}_S$. It must satisfy Snell's Law, and it must be a unit vector (by definition). This is adequate to obtain the expression

$$\hat{\mathbf{n}}_t(\mathbf{x},t) = \frac{n_i}{n_t}\hat{\mathbf{n}}_i + \Gamma(\mathbf{x},t)\,\hat{\mathbf{n}}_S(\mathbf{x},t) \tag{59}$$

with the function Γ defined as

$$\begin{aligned}\Gamma(\mathbf{x},t) \quad &\equiv \quad \frac{n_i}{n_t}\hat{\mathbf{n}}_i\cdot\hat{\mathbf{n}}_S(\mathbf{x},t) \\ &\pm \quad \left\{1-\left(\frac{n_i}{n_t}\right)^2 |\hat{\mathbf{n}}_i\times\hat{\mathbf{n}}_S(\mathbf{x},t)|^2\right\}^{1/2} \ .\end{aligned} \tag{60}$$

The plus sign is used in Γ when $\hat{\mathbf{n}}_i\cdot\hat{\mathbf{n}}_S < 0$, and the minus sign is used when $\hat{\mathbf{n}}_i\cdot\hat{\mathbf{n}}_S > 0$.

6.2 Fresnel Reflectivity and Transmissivity

Accompanying the process of reflection and transmission through the interface is a pair of coefficients that describe their efficiency. The reflectivity R and transmissivity T are related by the constraint that no light is lost at the interface. This leads to the relationship

$$R + T = 1 \ . \tag{61}$$

The derivation of the expressions for R and T is based on the electromagnetic theory of dielectrics. We will not carry out the derivations, but merely write down the solution

$$R(\hat{\mathbf{n}}_i,\hat{\mathbf{n}}_r) = \frac{1}{2}\left\{\frac{\sin^2(\theta_t-\theta_i)}{\sin^2(\theta_t+\theta_i)} + \frac{\tan^2(\theta_t-\theta_i)}{\tan^2(\theta_t+\theta_i)}\right\} \tag{62}$$

Figure 24 is a plot of the reflectivity for rays of light traveling down onto a water surface as a function of the angle of incidence to the surface. The plot extends from a grazing angle of 0 degrees to perpendicular incidence at 90 degrees. As should be clear, variation of the reflectivity across an image is an important source of the "texture" or feel of water. Notice that reflectivity is a function of the angle of incidence relative to the wave normal, which in turn is directly related to the slope of the surface. So we can expect that a strong contributor to the texture of water surface is the pattern of slope, while variation of the wave height serves primarily as a wave hiding mechanism. This is the quantitative explanation of why the

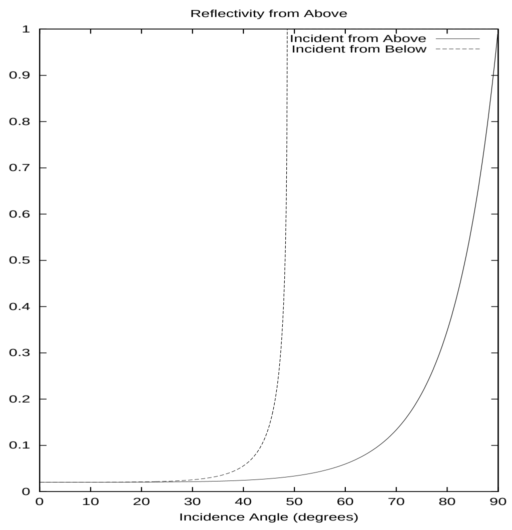

Figure 24: Reflectivity for light coming from the air down to the water surface, as a function of the angle of incidence of the light.

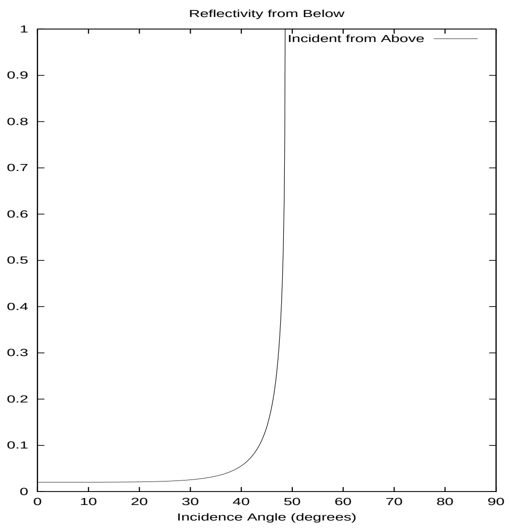

Figure 25: Reflectivity for light coming from below the water surface, as a function of the angle of incidence of the light.

surface slope more closely resembles rendered water than the wave height does, as we saw in the previous section when discussing figure 10.

When the incident ray comes from below the water surface, there are important differences in the reflectivity and transmissivity. Figure 25 shows the reflectivity as a function of incidence angle again, but this time for incident light from below. In this case, the reflectivity reaches unity at a fairly large angle, near 41 degrees. At incidence angles below that, the reflectivity is one and so there is no transmission of light through the interface. This phenomenon is *total internal reflection*, and can be seen just by swimming around in a pool. The angle at which total internal reflection begins is called Brewster's angle, and is given by, from Snell's Law,

$$\sin\theta_i^B = \frac{n_t}{n_i} = 0.75 \tag{63}$$

or $\theta_i^B = 48.6\,\mathrm{deg}$. In our plots, this angle is $90 - \theta_i^B = 41.1\,\mathrm{deg}$.

6.3 Building a Shader for Renderman

From the discussion so far, one of the most important features a rendering must emulate is the textures of the surface due to the strong slope-dependence of reflectivity and transmissivity. In this section we construct a simple Renderman-compliant shader using just these features. Readers who have experience with shaders will know how to extend this one immediately.

The shader exploits that fact that the Renderman interface already provides a built-in Fresnel quantity calculator, which provides R, T, $\hat{\mathbf{n}}_r$, and $\hat{\mathbf{n}}_t$ using the surface normal, incident direction vector, and index of refraction. The shader for the air-to-water case is as follows:

```
surface watercolorshader(
    color upwelling = color(0, 0.2, 0.3);
    color sky       = color(0.69,0.84,1);
    color air       = color(0.1,0.1,0.1);
    float nSnell    = 1.34;
    float Kdiffuse  = 0.91;
    string envmap   = "";
                        )
{
  float reflectivity;
  vector nI = normalize(I);
  vector nN = normalize(Ng);
  float costhetai = abs(nI . nN);
  float thetai = acos(costhetai);
  float sinthetat = sin(thetai)/nSnell;
  float thetat = asin(sinthetat);
  if(thetai == 0.0)
  {
    reflectivity = (nSnell - 1)/(nSnell + 1);
    reflectivity = reflectivity * reflectivity;
  }
  else
  {
    float fs = sin(thetat - thetai)
             / sin(thetat + thetai);
    float ts = tan(thetat - thetai)
             / tan(thetat + thetai);
    reflectivity = 0.5 * ( fs*fs + ts*ts );
  }
  vector dPE = P-E;
  float dist = length(dPE) * Kdiffuse;
  dist = exp(-dist);

  if(envmap != "")
  {
    sky = color environment(envmap, nN);
  }
  Ci =  dist * ( reflectivity * sky
               + (1-reflectivity) * upwelling )
     +  (1-dist)* air;
}
```

There are two contributions to the color: light coming downward onto the surface with the default color of the sky, and light coming upward from the depths with a default color. This second term will be discussed in the next section. It is important for incidence angles that are high in the sky, because the reflectivity is low and transmissivity is high.

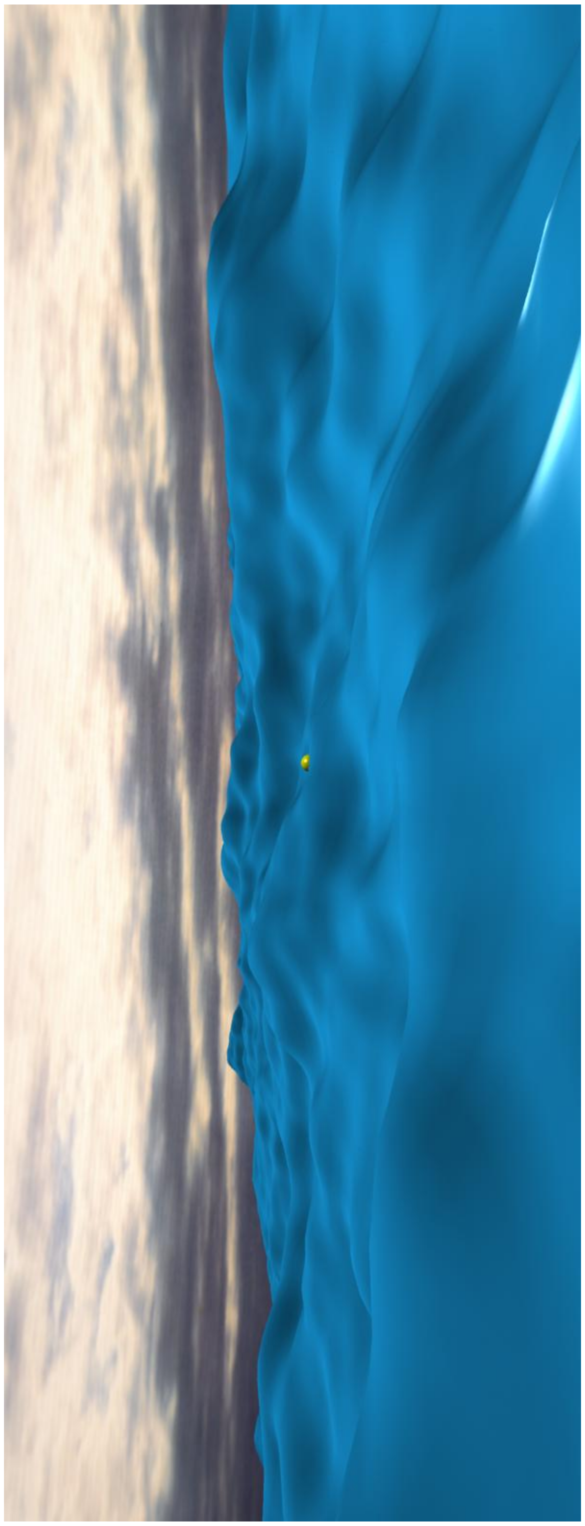

Figure 26: Simulated water surface with a generic plastic surface shader. Rendered with BMRT.

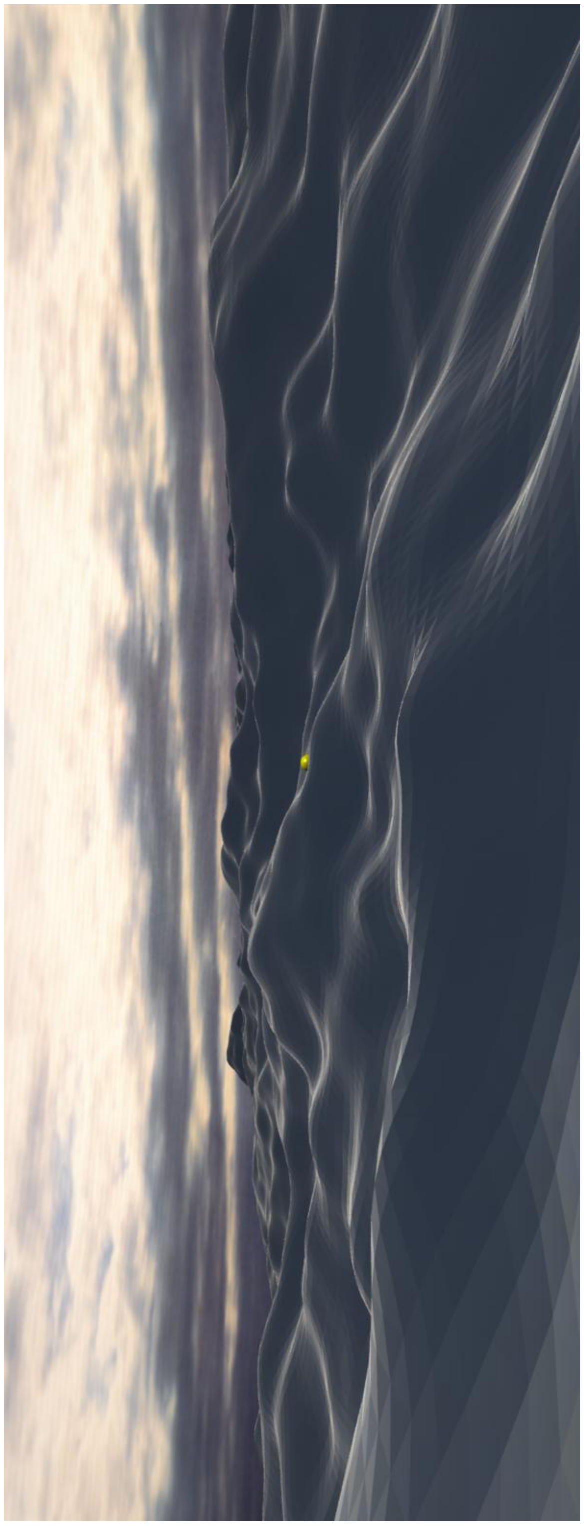

Figure 27: Simulated water surface with a realistic surface shader. Rendered with BMRT.

This shader was used to render the image in figure 27 using the BMRT raytrace renderer. For reference, the exact same image has been rendered in 26 with a generic plastic shader. Note that the realistic water shader tends to highlight the tops of the the waves, where the angle of incidence is nearly 90 degrees grazing and the reflectivity is high, while the sides of the waves are dark, where angle of incidence is nearly 0 that the reflectivity is low.

7 Water Volume Effects

The previous section was devoted to a discussion of the optical behavior of the surface of the ocean. In this section we focus on the optical behavior of the water volume below the surface. We begin with a discussion of the major optical effects the water volume has on light, followed by an introduction to color models researchers have built to try to connect the ocean color on any given day to underlying biological and physical processes. These models are built upon many years of in-situ measures off of ships and peers. Finally, we discuss two important effects, caustics and sunbeams, that sometimes are hard to grasp, and which produce beautiful images when properly simulated.

7.1 Scattering, Transmission, and Reflection by the Water Volume

In the open ocean, light is both scattering and absorbed by the volume of the water. The sources for these events are of three types: water molecules, living and dead organic matter, and non-organic matter. In most oceans around the world, away from the shore lines, absorption is a fairly even mixture of water molecules and organic matter. Scattering is dominated by organic matter however.

To simulate the processes of volumetric absorption and scattering, there are five quantities that are of interest: absorption coefficient, scattering coefficient, extinction coefficient, diffuse extinction coefficient, and bulk reflectivity. All of these coefficients have units of inverse length, and represent the exponential rate of attenuation of light with distance through the medium. The absorption coefficient a is the rate of absorption of light with distance, the scattering coefficient b is the rate of scattering with length, the extinction coefficient c is the sum of the two previous ones $c = a + b$, and the diffuse extinction coefficient K describes the rate of loss of intensity of light with distance after taking into account both absorption and scattering processes. The connection between K and the other parameters is not completely understood, in part because there are a variety of ways to define K in terms of operational measurements. Different ways change the details of the dependence. However, there is a condition called the *asymptotic* limit at very deep depths in the water, at which all operational definitions of K converge to a single value. This asymptotic value of K has been modeled in a variety of ways. There is a mathematically precise result that the ratio K/c depends only on b/c, the single scatter albedo, and some details of the angular distribution of individual scattering distributions. Figure 28 is an example of a model of K/c for reasonable water conditions. Models have been generated for the color dependence of K, most notably by Jerlov. In 1990, Austin and Petzold performed a revised analysis of spectral models, including new data, to produce refined models of K as a function of color. For typical visible light conditions in the ocean, K ranges in value from 0.03/meter to 0.1/meter. It is generally true that $a < K < c$.

One way to interpret these quantities for a simulation of water volume effects is as follows:

1. A ray of sunlight enters the water with intensity I (after losing some intensity to Fresnel transmission). Along a path underwater of a length s, the intensity at the end of the path is $I\exp(-cs)$, i.e. the ray of direct sunlight is attenuated as fast a possible.

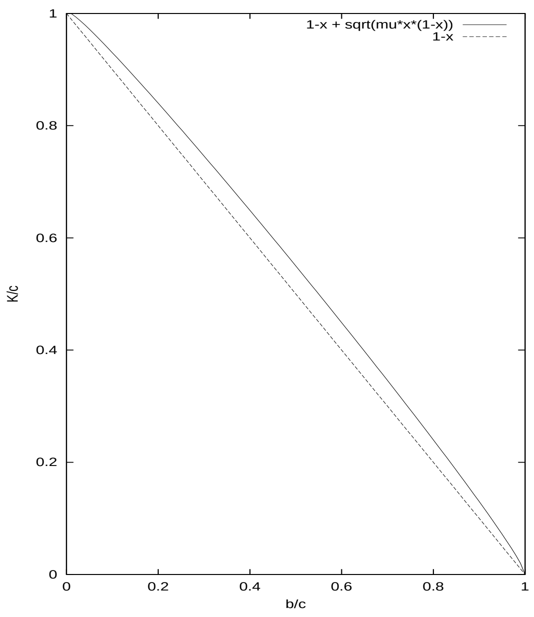

Figure 28: Dependence of the Diffuse Extinction Coefficient on the Single Scatter Albedo, normalized to the extinction.

2. Along the path through the water, a fraction of the ray is scattered into a distribution of directions. The strength of the scattering per unit length of the ray is b, so the intensity is proportional to $bI\exp(-cs)$.

3. The light that is scattered out of the ray goes through potentially many more scattering events. It would be nearly impossible to track all of them. However, the sum whole outcome of this process is to attenuate the ray along the path from the original path to the camera as $bI\exp(-cs)\exp(-Ks_c)$, where s_c is the distance from the scatter point in the ocean to the camera.

A fifth quantity of interest for simulation is the bulk reflectivity of the water volume. This is a quantity that is intended to allow us to ignore the details of what is going on, treat the volume as a Lambertian reflector, and compute a value for bulk reflectivity. That number is sensitive to many factors, including wave surface conditions, sun angle, water optical properties, and details of the angular spread. Nevertheless, values of reflectivity around 0.04 seem to agree well with data.

7.2 The Underwater POV: Refracted Skylight, Caustics, and Sunbeams

Now that we have underwater optical properties at hand, we can look at two important phenomena in the ocean: caustics and sunbeams.

7.2.1 Caustics

Caustics, in this context, are a light pattern that is formed on surfaces underwater. Because the water surface is not flat, groups of

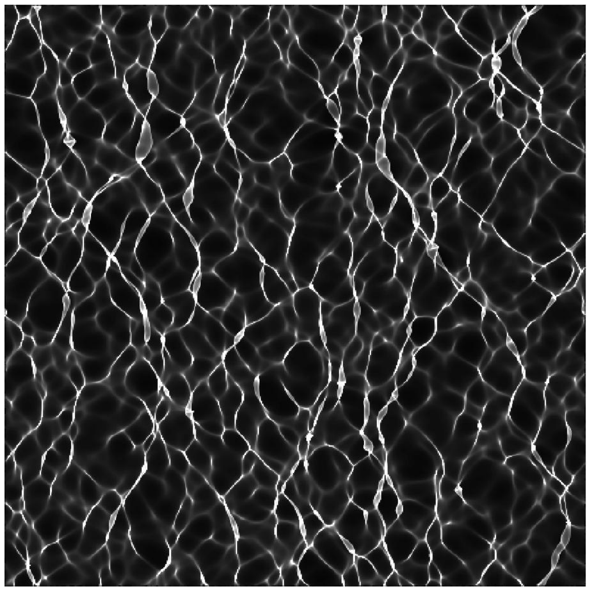

Figure 29: Rendering of a caustic pattern at a shallow depth (5 meters) below the surface.

light rays incident on the surface are either focussed or defocussed. As a result, a point on a fictitious plane some depth below the ocean surface receives direct sunlight from several different positions on the surface. The intensity of light varies due to the depth, original contrast, and other factors. For now, lets write the intensity of the pattern as $I = Ref\ I_0$, with I_0 as the light intensity just above the water surface. The quantity Ref is the scaling factor that varies with position on the fictitious plane due to focussing and defocussing of waves, and is called a *caustic pattern*. Figure 29 shows an example of the caustic pattern Ref. Notice that the caustic pattern exhibits filaments and ring-like structure. At a very deep depth, the caustic pattern is even more striking, as shown in figure 30.

One of the important properties of underwater light that produce caustic patterns is conservation of flux. This is actually a simple idea: suppose a small area on the ocean surface has sunlight passing through it into the water, with intensity I at the surface. As we map that area to greater depths, the amount of area within it grows or shrinks, but most likely grows depending on whether the area is focussed or defocussed. The intensity at any depth within the water is proportional to inverse of the area of the projected region. Another way of saying this is that if a bundle of light rays diverges, their intensities are reduced to keep the product of intensity time area fixed.

Simulated caustic patterns can actually be compared (roughly) with real-world data. In a series of papers published throughout the 1970's, 1980's, and into the 1990's, Dera and others collected high-speed time series of light intensity[21]. As part of this data collection and analysis project, the data was used to generate a probability distribution function (PDF) for the light intensity. Figure 31 shows two PDFs taken from one of Dera's papers. The two PDF's were collected for different surface roughness conditions: rougher water tended to suppress more of the high magnitude fluctuations in intensity.

Figure 32 shows the pdf at two depths from a simulation of the ocean surface. These two sets do not match Dera's measurements because of many factors, but most importantly because we have not simulated the environmental conditions and instrumentation in Dera's experiments. Nevertheless, the similarity of figure 32 with

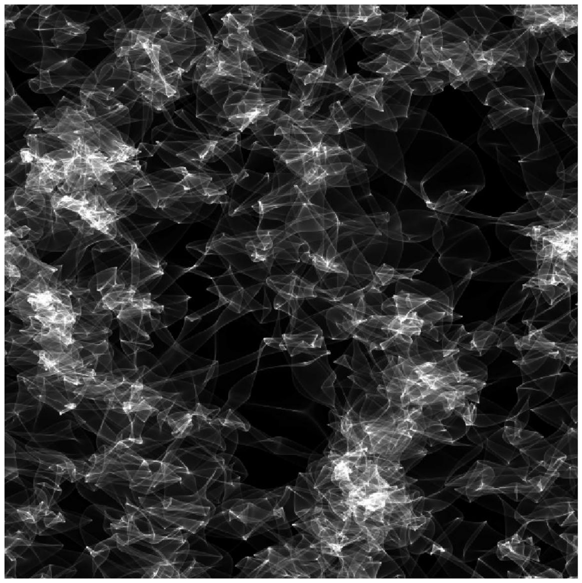

Figure 30: Rendering of a caustic pattern at great depth (100 meters) below the surface.

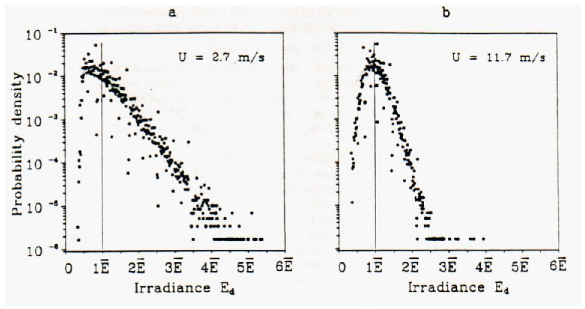

Figure 31: PDF's as measured by Dera in reference [21].

Dera's data is an encouraging point of information for the realism of the simulation.

7.2.2 Godrays

Underwater sunbeams, also called godrays, have a very similar origin to caustics. Direct sunlight passes into the water volume, focussed and defocussed at different points across the surface. As the rays of light pass down through the volume, some of the light is scattered in other directions, and a fraction arrives at the camera. The accumulated pattern of scattered light apparent to the camera are the godrays. So, while caustics are the pattern of direct sunlight that penetrates down to the floor of a water volume, sunbeams are scattered light coming from those shafts of direct sunlight in the water volume. Figure 33 demonstrates sunbeams as seen by a camera looking up at it.

References

[1] Jos Stam, "Stable Fluids," *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp 121-128, (1999).

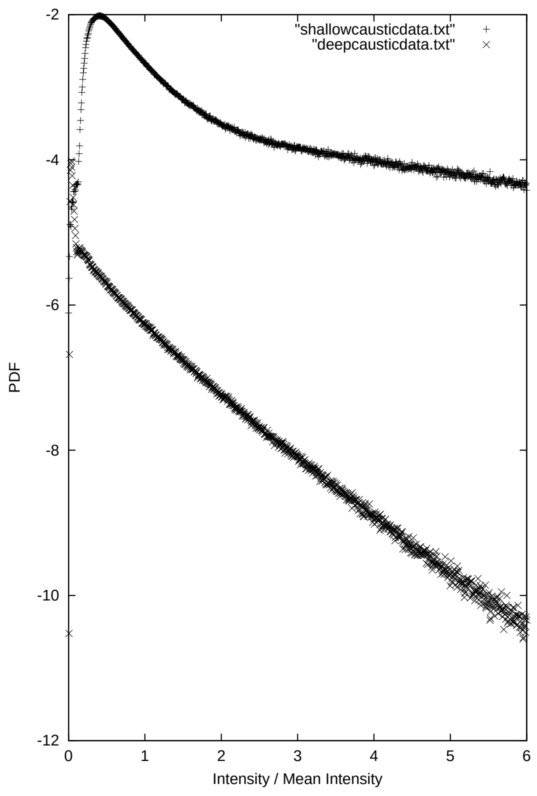

Figure 32: Computed Probability Density Function for light intensity fluctuations in caustics. (upper curve) shallow depth of 2 meters; (lower curve) deep depth of 10 meters.

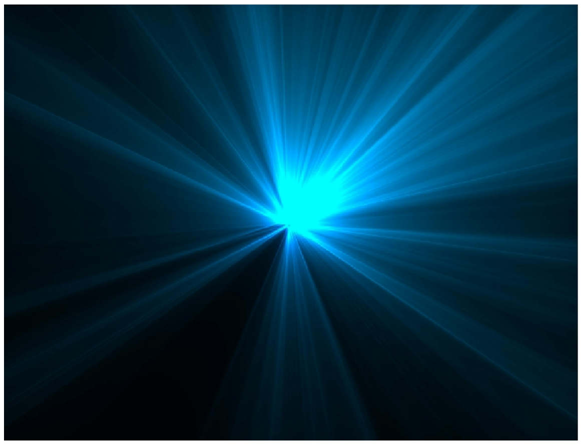

Figure 33: Rendering of sunbeams, or godrays, as seen looking straight up at the light source.

[2] Nick Foster and Ronald Fedkiw, "Practical animation of liquids," *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp 23-30, (2001).

[3] B.A. Cheeseman and C.P.R. Hoppel,"SIMULATING THE BALLISTIC IMPACT OF COMPOSITE STRUCTURAL ARMOR," http://www.asc2002.com/summaries/h/HP-08.pdf

[4] Simon Premoze, "Particle-Based Simulation of Fluids," *Eurographics 2003*, Vol 22, No. 3.

[5] Jeff Odien, "On the Waterfront", Cinefex, No. 64, p 96, (1995)

[6] Ted Elrick, "Elemental Images", Cinefex, No. 70, p 114, (1997)

[7] Kevin H. Martin, "Close Contact", Cinefex, No. 71, p 114, (1997)

[8] Don Shay, "Ship of Dreams", Cinefex, No. 72, p 82, (1997)

[9] Kevin H. Martin, "Virus: Building a Better Borg", Cinefex, No. 76, p 55, (1999)

[10] Simon Premože and Michael Ashikhmin, "Rendering Natural Waters," Eighth Pacific Conference on Computer Graphics and Applications, October 2000.

[11] Gary A. Mastin, Peter A. Watterger, and John F. Mareda, "Fourier Synthesis of Ocean Scenes", IEEE CG&A, March 1987, p 16-23.

[12] Alain Fournier and William T. Reeves, "A Simple Model of Ocean Waves", Computer Graphics, Vol. 20, No. 4, 1986, p 75-84.

[13] Darwyn Peachey, "Modeling Waves and Surf", Computer Graphics, Vol. 20, No. 4, 1986, p 65-74.

[14] Blair Kinsman, *Wind Waves, Their Generation and Propagation on the Ocean Surface*, Dover Publications, 1984.

[15] S. Gran, *A Course in Ocean Engineering, Developments in Marine Technology No. 8*, Elsevier Science Publishers B.V. 1992. See also `http://www.dnv.no/ocean/bk/grand.htm`

[16] Dennis B. Creamer, Frank Henyey, Roy Schult, and Jon Wright, "Improved Linear Representation of Ocean Surface Waves." J. Fluid Mech, **205**, pp. 135-161, (1989).

[17] Seibert Q. Duntley, "Light in the Sea," J. Opt. Soc. Am., **53**,2, pg 214-233, 1963.

[18] Curtis D. Mobley, *Light and Water: Radiative Transfer in Natural Waters*, Academic Press, 1994.

[19] *Selected Papers on Multiple Scattering in Plane-Parallel Atmospheres and Oceans: Methods*, ed. by George W. Kattawar, SPIE Milestones Series, **MS 42**, SPIE Opt. Eng. Press., 1991.

[20] R.W. Austin and T. Petzold, "Spectral Dependence of the Diffuse Attenuation Coefficient of Light in Ocean Waters: A Reexamination Using New Data," *Ocean Optics X*, Richard W. Spinrad, ed., SPIE **1302**, 1990.

[21] Jerzy Dera, Slawomir Sagan, Dariusz Stramski, "Focusing of Sunlight by Sea Surface Waves: New Measurement Results from the Black Sea," *Ocean Optics XI*, SPIE Proceedings, 1992.

[22] Jerry Tessendorf, “Interactive Water Surfaces,” *Game Programming Gems 4* , ed. Andrew Kirmse, Charles River Media, (2004).

[23] Stéphan T. Grilli home page, `http://131.128.105.170/ grilli/`.

[24] AROSS - Airborne Remote Optical Spotlight System, `http://www.aross.arete-dc.com`

8 Appendix: Sample code for interactive water surfaces

The code listed in this appendix is a working implementation of the iWave algorithm. As listed below, *iwave_paint* looks like a crude paint routine. The user can paint in two modes. When *iwave_paint* starts up, it begins running an iWave simulation on a small grid (200x200 with the settings listed). This grid is small enough that the iWave simulation runs interactively on cpus around 1 GHz and faster. The running of the simulation is not apparent since there are no disturbing waves at start up. The interface looks like figure 34. In the default mode at start up, the painting generates obstructions that show up in black and block water waves. Figure 35 shows an example obstruction that has been painted. Hitting the ’s’ key changes the painting mode to painting a source disturbance on the water surface. As you paint the disturbance, *iwave_paint* propagates the disturbance, reflecting off of any obstructions you may have painted. Figure 36 shows a frame after source has been painted inside the obstructed and allows a brief period of time to propagate inside the obstruction and exit, creating a diffraction pattern at the mouth of the obstruction.

A few useful keyboard options:

o Puts the paint mode into obstruction painting. This may be selected at any time.

s Puts the paint mode into source painting. This may be selected at any time.

```
//-----------------------------------------------
//
//  iwave_paint
//
//  demonstrates the generation and interaction of
//  waves around objects by allowing the user to
//  paint obstructions and source, and watch iwave
//  propagation.
//
//  author: Jerry Tessendorf
//          jerry@finelightvisualtechnology.com
//  August, 2003
//
//  This software is in the public domain.
//
//------------------------------------------------

//------------------------------------------------
//
//  usage:
//
//  iwave_paint is an interactive paint program
//  in which the user paints on a water surface and
//  the waves evolve and react with obstructions
//  in the water.
//
//  There are two paint modes.  Typing ’o’ puts the
//  program in obstruction painting mode. When you
//  hold down the left mouse button and paint, you
//  will see a black obstruction painted.  This
//  obstruction may be any shape.
//
//  Typing ’s’ puts the program in source painting
//  mode.  Now painting with the left mouse button
//  down generates a source disturbance on the water
//  surface.  The waves it produces evolve if as you
//  continue to paint.  The waves bounce of off any
//  obstructions that have been painted or are
//  subsequently painted.
//
//  Typing ’b’ clears all obstructions and waves.
//
//  Typing ’=’ and ’-’ brightens and darkens the display
//  of the waves.
//
//  Pressing the spacebar starts and stops the wave
//  evolution. While the evolution is stopped, you
//  can continue painting obstructions.
//
//
//  This code was written and runs under Linux. The
//  compile command is
//
//  g++ iwave_paint.C -O2 -o iwave_paint -lglut -lGL
//
//
//------------------------------------------------


#include <cmath>
```

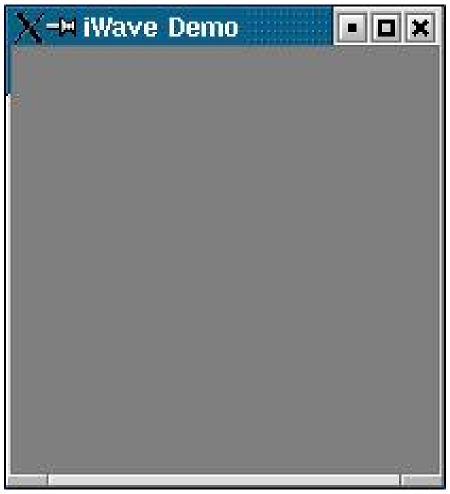

Figure 34: The *iwave_paint* window at startup.

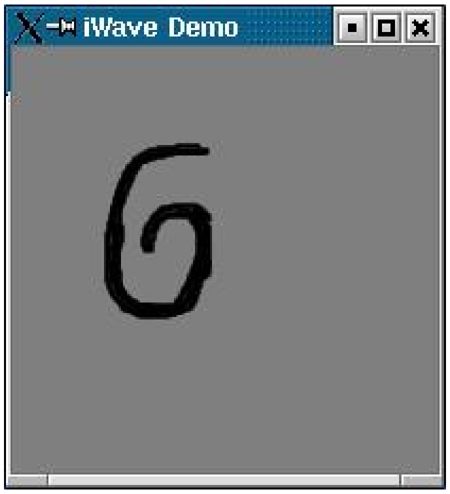

Figure 35: The *iwave_paint* window with obstruction painted.

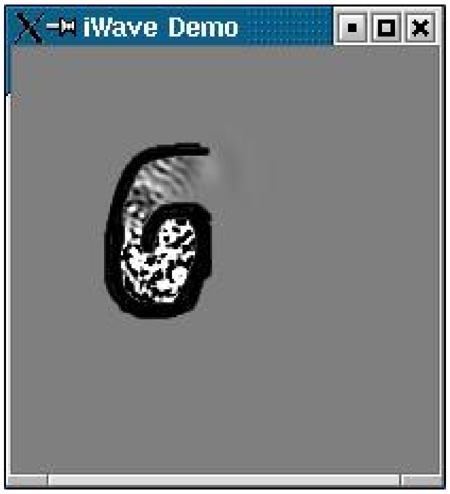

Figure 36: The *iwave_paint* window after painting source inside the obstruction and letting it propagate out.

```
#ifdef __APPLE__
  #include <GLUT/glut.h>
#else
  #include <GL/gl.h>   // OpenGL itself.
  #include <GL/glu.h>  // GLU support library.
  #include <GL/glut.h> // GLUT support library.
#endif

#include <iostream>

using namespace std;

int iwidth, iheight, size;
float *display_map;
float *obstruction;
float *source;

float *height;
float *previous_height;
float *vertical_derivative;

float scaling_factor;
float kernel[13][13];

int paint_mode;
enum{ PAINT_OBSTRUCTION, PAINT_SOURCE };

bool regenerate_data;
bool toggle_animation_on_off;

float dt, alpha, gravity;

float obstruction_brush[3][3];
float source_brush[3][3];

int xmouse_prev, ymouse_prev;


//--------------------------------------------------------
//
//  Initialization routines
//
//
// Initialize all of the fields to zero
void Initialize( float *data, int size, float value )
{
   for(int i=0;i<size;i++ ) { data[i] = value; }
}

// Compute the elements of the convolution kernel
void InitializeKernel()
{
    double dk = 0.01;
    double sigma = 1.0;
    double norm = 0;

    for(double k=0;k<10;k+=dk)
    {
        norm += k*k*exp(-sigma*k*k);
    }

    for( int i=-6;i<=6;i++ )
    {
for( int j=-6;j<=6;j++ )
{
   double r = sqrt( (float)(i*i + j*j) );
           double kern = 0;
   for( double k=0;k<10;k+=dk)
   {
       kern += k*k*exp(-sigma*k*k)*j0(r*k);
   }
           kernel[i+6][j+6] = kern / norm;
}
    }
}


void InitializeBrushes()
{
   obstruction_brush[1][1] = 0.0;
   obstruction_brush[1][0] = 0.5;
   obstruction_brush[0][1] = 0.5;
   obstruction_brush[2][1] = 0.5;
   obstruction_brush[1][2] = 0.5;
   obstruction_brush[0][2] = 0.75;
   obstruction_brush[2][0] = 0.75;
   obstruction_brush[0][0] = 0.75;
   obstruction_brush[2][2] = 0.75;

   source_brush[1][1] = 1.0;
   source_brush[1][0] = 0.5;
   source_brush[0][1] = 0.5;
   source_brush[2][1] = 0.5;
   source_brush[1][2] = 0.5;
   source_brush[0][2] = 0.25;
   source_brush[2][0] = 0.25;
   source_brush[0][0] = 0.25;
   source_brush[2][2] = 0.25;
}

void ClearObstruction()
{
    for(int i=0;i<size;i++ ){ obstruction[i] = 1.0; }
}

void ClearWaves()
{
    for(int i=0;i<size;i++ )
    {
      height[i] = 0.0;
      previous_height[i] = 0.0;
      vertical_derivative[i] = 0.0;
   }
}

//----------------------------------------------------

void ConvertToDisplay()
{
   for(int i=0;i<size;i++ )
   {
      display_map[i] = 0.5*( height[i]/scaling_factor + 1.0 )*obstruction[i];
   }
}

//----------------------------------------------------
//
//  These two routines,
//
//    ComputeVerticalDerivative()
//    Propagate()
//
//  are the heart of the iWave algorithm.
//
//  In Propagate(), we have not bothered to handle the
//  boundary conditions.  This makes the outermost
//  6 pixels all the way around act like a hard wall.
//

void ComputeVerticalDerivative()
{
   // first step:  the interior
   for(int ix=6;ix<iwidth-6;ix++)
   {
       for(int iy=6;iy<iheight-6;iy++)
{
   int index = ix + iwidth*iy;
   float vd = 0;
   for(int iix=-6;iix<=6;iix++)
   {
      for(int iiy=-6;iiy<=6;iiy++)
      {
  int iindex = ix+iix + iwidth*(iy+iiy);
         vd += kernel[iix+6][iiy+6] * height[iindex];
      }
   }
   vertical_derivative[index] = vd;
}
   }
}

void Propagate()
{
   // apply obstruction
   for( int i=0;i<size;i++ ) { height[i] *= obstruction[i]; }

   // compute vertical derivative
   ComputeVerticalDerivative();

   // advance surface
   float adt = alpha*dt;
   float adt2 = 1.0/(1.0+adt);
   for( int i=0;i<size;i++ )
   {
      float temp = height[i];
      height[i] = height[i]*(2.0-adt)-previous_height[i]-gravity*vertical_derivative[i];
      height[i] *= adt2;
      height[i] += source[i];
      height[i] *= obstruction[i];
      previous_height[i] = temp;
      // reset source each step
      source[i] = 0;
   }
}


//------------------------------------------
//
//  Painting and display code
//

void resetScaleFactor( float amount )
{
  scaling_factor *= amount;
}

void DabSomePaint( int x, int y )
{
  int xstart = x - 1;
  int ystart = y - 1;
  if( xstart < 0 ){ xstart = 0; }
  if( ystart < 0 ){ ystart = 0; }

  int xend = x + 1;
  int yend = y + 1;
  if( xend >= iwidth ){ xend = iwidth-1; }
  if( yend >= iheight ){ yend = iheight-1; }


  if( paint_mode == PAINT_OBSTRUCTION )
  {
     for(int ix=xstart;ix <= xend; ix++)
     {
        for( int iy=ystart;iy<=yend; iy++)
 {
           int index = ix + iwidth*(iheight-iy-1);
   obstruction[index] *= obstruction_brush[ix-xstart][iy-ystart];
 }
     }
  }
  else if( paint_mode == PAINT_SOURCE )
  {
     for(int ix=xstart;ix <= xend; ix++)
     {
        for( int iy=ystart;iy<=yend; iy++)
 {
           int index = ix + iwidth*(iheight-iy-1);
   source[index] += source_brush[ix-xstart][iy-ystart];
 }
     }
  }
  return;
}

//----------------------------------------------------
//
//  GL and GLUT callbacks
//
//----------------------------------------------------


void cbDisplay( void )
{
  glClear(GL_COLOR_BUFFER_BIT );
  glDrawPixels( iwidth, iheight, GL_LUMINANCE, GL_FLOAT, display_map );
  glutSwapBuffers();
}

// animate and display new result
void cbIdle()
{
  if( toggle_animation_on_off ) { Propagate(); }
  ConvertToDisplay();
  cbDisplay();
}


void cbOnKeyboard( unsigned char key, int x, int y )
{
  switch (key)
  {
     case '-': case '_':
     resetScaleFactor( 1.0/0.9 );
     regenerate_data = true;
     break;

     case '+': case '=':
     resetScaleFactor( 0.9 );
     regenerate_data = true;
     break;

     case ' ':
     toggle_animation_on_off = !toggle_animation_on_off;

     case 'o':
     paint_mode = PAINT_OBSTRUCTION;
     break;

     case 's':
     paint_mode = PAINT_SOURCE;
     break;

     case 'b':
     ClearWaves();
     ClearObstruction();
     Initialize( source, size, 0.0 );
     break;

     default:
     break;
  }
}

void cbMouseDown( int button, int state, int x, int y )
{
  if( button != GLUT_LEFT_BUTTON ) { return; }
  if( state != GLUT_DOWN ) { return; }
  xmouse_prev = x;
  ymouse_prev = y;
  DabSomePaint( x, y );
}


void cbMouseMove( int x, int y )
{
  xmouse_prev = x;
  ymouse_prev = y;
  DabSomePaint( x, y );
}



//---------------------------------------------------

int main(int argc, char** argv)
{
  // initialize a few variables
  iwidth = iheight = 200;
  size = iwidth*iheight;

  dt = 0.03;
  alpha = 0.3;
  gravity = 9.8 * dt * dt;

  scaling_factor = 1.0;
  toggle_animation_on_off = true;
```

```
   // allocate space for fields and initialize them
   height              = new float[size];
   previous_height     = new float[size];
   vertical_derivative = new float[size];
   obstruction         = new float[size];
   source              = new float[size];
   display_map         = new float[size];

   ClearWaves();
   ClearObstruction();
   ConvertToDisplay();
   Initialize( source, size, 0 );

   InitializeBrushes();

   // build the convolution kernel
   InitializeKernel();



   // GLUT routines
   glutInit(&argc, argv);

   glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
   glutInitWindowSize( iwidth, iheight );

   // Open a window
   char title[] = "iWave Demo";
   int Window_ID = glutCreateWindow( title );

   glClearColor( 1,1,1,1 );

   glutDisplayFunc(&cbDisplay);
   glutIdleFunc(&cbIdle);
   glutKeyboardFunc(&cbOnKeyboard);
   glutMouseFunc( &cbMouseDown );
   glutMotionFunc( &cbMouseMove );

   glutMainLoop();
   return 1;

};
```