Real-Time Synthesis and Rendering of Ocean Water

Jason L. Mitchell

ATI Research Technical Report, April 2005

(a) Low band of wave frequencies (b) Broad band of wave frequencies (c) Shallow water damping

Figure 1: Polygonal water surface displaced and shaded with GPU-synthesized height and normal maps. (a) Low frequencies used for displacement and shading. (b) Low frequencies used for displacement and all frequencies used for shading. (c) Damping of high frequencies closer to the camera due to shallow water and presence of vegetation.

Abstract

We present a multi-band Fourier domain approach to synthesizing and rendering deep-water ocean waves entirely on the graphics processor. Our technique begins by using graphics hardware to generate and animate a Fourier domain spectrum of ocean water. We subsequently use the graphics hardware to apply an IFFT to transform the spectrum into a realistic height map of ocean water in the spatial domain. As a result, this technique can be used to efficiently synthesize and render height maps and normal maps which are spatially periodic. GPU-synthesized low frequency height maps are used to displace geometry while broad spectrum GPU-synthesized waveforms are used to generate a normal map for shading. The model also allows for compositing of other waveforms such as wakes or eddies caused by objects interacting with the water. The primary contributions of this work are the use of the GPU to perform all synthesis and rendering steps as well as the multi-band approach which enables efficient simulation of the natural filtering of high frequencies at different points on the water surface due to depth variation or the presence of plant matter.

1 Introduction

Here we will present a multi-band Fourier domain approach to synthesizing and rendering deep-water ocean waves entirely on the graphics processor. This technique can be used to efficiently synthesize height maps and normal maps which are spatially periodic. Low-frequency height maps can be used to displace geometry while the full spectrum of the synthesized height maps can be used as a normal map for shading. This allows us to generate realistic water surfaces on demand rather than store precomputed maps or simply scroll repeating noise maps.

Our primary contributions are the use of the GPU to perform all synthesis and rendering and a multi-band approach which can be used to generate a low-frequency geometry displacement map and high frequency normal map for shading as well as approximating depth effects.

Even at low resolutions where the CPU, using SIMD extensions, could theoretically outperform the GPU, the migration of this algorithm is still a win for hardware accelerated rendering because the system-to-video memory transfer bottleneck is removed and the CPU cycles are freed up for other computations such as game logic.

We will begin by reviewing previous work in the field. After we have reviewed both spatial and Fourier domain techniques, we will describe the theory and practice of our synthesis algorithm in detail in section 3. We will describe our rendering algorithm in section 4. In section 5, we will discuss a technique for incorporating other waveforms such as ripples or wakes as well as a technique for synthesizing plausible waves for small bodies of water such as ponds which have shallow regions and plant matter that may damp out high frequency waves. In order to place this work in the context of the computing system as a whole, we will describe the performance of our technique in section 6. We will then conclude with a summary and a look at future work.

2 Previous work

Previous work in ocean water synthesis for computer graphics can be broken into two classes: *spatial domain* and *Fourier domain* approaches.

2.1 Spatial Domain

Max computed a height field composed of the superposition of several low amplitude sinusoids [Max81]. The frequencies of these waveforms were chosen so that they would repeat at a common point in time so that their motion could be looped and reused throughout an animation. For rendering, the waves were ray-traced

on a Cray 1 using up to two ray bounces.

Perlin applied noise to a finite set of spherical wave fronts and used the result to perturb the normals of a plane without displacing the surface in space [Perlin85]. This gave a reasonably convincing result at certain scales and has the advantage that it can be applied seamlessly to arbitrary surfaces.

Fournier and Reeves applied a large number of modifications to the Gerstner model to incorporate shore interactions such as wave refraction and breaking [Fournier86]. This model does modify the positions of the surface and even allows for overhangs in the case of wave breaking.

Peachy generates a height field by computing the superposition of several long-crested waveforms [Peachey86]. In addition to depth-dependent blending of a quadratic function with the underlying sinusoids to give a more realistic cycloidal choppy appearance, Peachy also accounted for other depth effects such as wave refraction and emission of particles to simulate spray.

Ts'o and Barsky generate a Beta-spline surface by wave tracing a small number of wave fronts to simulate wave refraction in shallow water [Ts'o87]. For illumination, reflective and refractive terms are blended based upon a Fresnel term computed at each pixel.

Thon views the height field as being composed of a main structure modeled as the superposition of 2D trochoids and a detail level modeled as perturbations caused by a 3D turbulence function [Thon00]. This technique begins by selecting a set of high-amplitude trochoids from the Pierson-Moskowitz spectrum and directly calculating their superposition in the spatial domain. This surface is further perturbed using a 3D turbulence function. Both the main structure and fine details can be animated to produce realistic motion and shading of ocean water height fields.

There have been several recent attempts to generate plausible real-time water surface animation on graphics hardware in the spatial domain. Schneider and Westermann use OpenGL evaluators and vertex shaders to combine two octaves of gradient noise to perturb a height field at interactive rates [Schneider01]. Isidoro et al compute geometry displacement as the superposition of four low-frequency sinusoids using hardware vertex shaders [Isidoro02]. Corresponding analytical normals are also computed in the vertex shader and tangent space normal maps are scrolled across the displaced surface to provide high frequency components. While both of these techniques produce water that may be useful in some interactive applications, they do not model real ocean water accurately enough to rival offline techniques.

Hinsinger et al have developed a technique which adaptively tessellates a grid of points and appropriately bandlimits a set of up to 60 spatial domain Gerstner waves for real-time rendering [Hinsinger02]. This tessellation and waveform superposition is performed on the CPU, which is the bottleneck of this approach. The mesh is uploaded to the GPU each frame. This technique supports the inclusion of arbitrary waveforms such as wakes, which is also supported by our technique.

While spatial domain approaches can have the advantages that only visible regions and frequencies of the ocean surface are evaluated for each rendered frame, it is not clear that these algorithms are readily adapted for hardware acceleration. In fact, it has been argued that agitated ocean water surfaces require too many spatial domain terms for acceptable performance [Thon00]. Fourier domain techniques seek to overcome this limitation by performing synthesis in the Fourier domain, thus avoiding the cost of explicitly computing the superposition of spatial domain sinusoids.

2.2 Fourier Domain

Fourier domain approaches, first introduced to the computer graphics community by Mastin et al, seek to synthesize ocean water by utilizing measured spectral properties of real ocean water [Mastin87]. Typically, some type of noise is transformed to the Fourier domain and then filtered according to known statistical models of ocean water. This Fourier domain description of the ocean water is then transformed to the spatial domain and treated as a height map which can be tiled seamlessly over a larger domain for offline or real-time rendering. One disadvantage of this approach relative to more costly spatial domain approaches is that, for high-altitude flyovers, it is obvious that the synthesized ocean water surface is composed of repeating tiles. For many scales of interactive application, however, this repetition is not apparent to the user.

Mastin transformed white noise from the spatial to the Fourier domain and filtered it with a Pierson-Moskowitz spectrum. The IFFT of this spectrum resulted in a realistic ocean water height map, which could be animated by appropriately shifting the phase in the Fourier domain each frame.

In a series of SIGGRAPH course notes on natural phenomena, Tessendorf describes a similar approach which has been applied to production rendering in films such as *Waterworld*, *Titanic* and others [Tessendorf99]. Tessendorf starts in the frequency domain and uses the Phillips spectrum rather than the Pierson-Moskowitz spectrum. Tessendorf also proposes a variety of enhancements, allowing him to tune the Phillips spectrum to influence the direction, speed and parallel nature of the resulting wave field. Tessendorf also proposes post-processing of the synthesized height map to increase choppiness since the IFFT, by definition, does not directly result in trochoidal shapes.

Jensen and Goliáš describe results of adapting many of Tessendorf's techniques to real-time, including wave generation, choppiness, foam, spray, caustics and even godrays [Jensen01]. Additionally, Jensen and Goliáš use a CPU implementation of the Navier-Stokes Equations to compute a normal map to represent turbulent flow around objects in the water. This work is the most similar to ours, except that all computations are performed on the CPU, significantly limiting the water height map and normal map resolutions. Although Jensen and Goliáš use separate CPU-generated frequency bands for geometry and normal maps, they do not implement a multi-band approach to controllably simulate the damping of high frequencies due to depth variation and plant matter as we will describe in Section 5 of this paper.

Most recently, Lanza implemented a similar scheme in which the IFFT is computed on the CPU and used to generate a dynamically displaced mesh with corresponding tangent space basis vectors [Lanza04]. Geometric complexity is managed with a quad-tree scheme. As in [Isidoro02], a tangent space normal map is used when shading this surface. Unlike our technique, however, this normal map does not contain the underlying water signal and must be rotated into world space per pixel.

3 Wave Generation and Animation

It is desirable to be able to generate ocean water height fields on the fly in order to conserve memory on low-memory systems such as game consoles. With Fourier domain techniques, it is possible to synthesize an ocean water height field at an arbitrary point in time and on current graphics processors it is possible to do this quickly enough to be useful in an interactive application. The technique described here is built upon models of deep ocean water with no shore interactions but our experiments indicate that it is possible to composite our synthesized deep-water waveforms with other simulated or pre-computed waveforms to generate plausible shore interactions in many cases. In Section 3.4, we will demonstrate the ability to composite arbitrary waveforms such as boat wake with the synthesized waves.

3.1 Synthesis Theory

We will now present a brief review the theory of Fourier domain ocean wave synthesis and animation. For consistency, we will use Tessendorf's notation, which will make use of the following mathematical symbols:

Symbol	Description
k	wave number
$\mathbf{k}$	wave vector (k_x, k_y)
T	Period of wave
λ	wavelength
h	Height of water
$\mathbf{x}$	Spatial position of simulation point
t	time
g	gravitational constant
P_k	Phillips spectrum
ξ	Ordinary independent draw from a Gaussian random number generator with mean a mean of zero and standard deviation of one
L	Largest possible wave arising from a given wind speed
ω	angular frequency
w	wind direction

Fundamentally, we wish to model the ocean surface as a height field $h(\mathbf{x}, t)$ which is a sum of sinusoids with time-dependent amplitudes. The height of the water at location $\mathbf{x}$ at time t is:

$$h(\mathbf{x}, t) = \sum_{\mathbf{k}} \tilde{h}(\mathbf{k}, t) e^{i\mathbf{k}\mathbf{x}} \tag{1}$$

where $\mathbf{k}$ is a 2D vector with components (k_x, k_y), $k_x = 2\pi n / L_x$, $k_y = 2\pi m / L_y$ and n and m are integers with bounds $-N/2 \leq n < N/2$ and $-M/2 \leq m < M/2$. The IFFT, which is the workhorse of this algorithm will generate a height field at discrete points $\mathbf{x} = (nL_x/N, mL_y/M)$ which we will use to construct ocean water.

Since we wish to construct a repeatable tile of displaced ocean water, it is convenient to construct a frequency-domain representation and take its inverse Fourier transform since, by definition, this produces an appropriate spatially periodic signal (height field). With the addition of floating-point pixel pipelines and floating-point textures to commodity graphics cards, it is possible to implement classic image processing operations such as the FFT on the GPU [Moreland03] [Mitchell03]. With this fundamental building block, we can apply Fourier synthesis techniques to hardware accelerated ocean wave synthesis and animation.

We proceed directly from Tessendorf's work and define a set of complex Fourier domain amplitudes and their initial phase values for our wave height field at time zero:

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

The Phillips Spectrum, P_h, is a popular model used for wind-driven ocean waves:

$$P_h(\mathbf{k}) = A\frac{\exp(-1/(kL)^2)}{k^4}|\hat{\mathbf{k}} \cdot w|^2 \tag{3}$$

where $L = V^2/g$ is the largest possible wave arising from a continuous wind speed V and A is a numeric constant which globally affects the wave heights. It is possible to tune this model for the desired look of a given rendering application and we have enabled interactive tweaking of many of the terms in this equation in our real-time application. The exponent in the denominator can be changed, for example, to decrease the spread of wave propagation directions and make the waves appear more parallel in the spatial domain. Additionally, the $|\hat{k} \cdot w|^2$ term is responsible for eliminating waves moving perpendicular to the wind direction. As Tessendorf points out, it is also helpful to suppress waves smaller than a small wavelength $l \ll L$ and modify the Phillips spectrum by multiplying by $exp(-k^2l^2)$. Tessendorf also points out that it is possible to change the exponent in the denominator to other powers such as 6 to increase the apparent directionality of the synthesized ocean waves. This is especially useful when one wishes to synthesize waves that appear to be approaching a shoreline.

Unlike sound waves in air, water waves are *dispersive*, which means that their velocity depends on their wavelength. In deep ocean water, ignoring small-scale capillary waves, for a wave of pulsation $\omega = 2\pi/T$ and wave number $k = 2\pi/\lambda$, the relation between ω and k is $\omega = \sqrt{gk}$. Given this dispersion relation $\omega(k)$, the Fourier domain amplitudes of our wave field at time t can be expressed as:

$$\tilde{h}(\mathbf{k}, t) = \tilde{h}_0(\mathbf{k})e^{i\omega(k)t} + \tilde{h}_0^*(-\mathbf{k})e^{-i\omega(k)t} \tag{4}$$

This relation preserves the complex conjugation property $\tilde{h}^*(\mathbf{k}, t) = \tilde{h}(-\mathbf{k}, t)$ and is convenient because it will allow us to compute $\tilde{h}(\mathbf{k}, t)$ and hence $h(\mathbf{x}, t)$ via the IFFT on demand at any time t without having to compute it at any other time.

Now that we have reviewed some of the theory of Fourier domain ocean water synthesis, we will describe our implementation which enables us to migrate all of the synthesis to the graphics processor.

3.2 GPU Implementation

While it is important to be familiar with the theoretical underpinnings of this technique, very little of the above math must be applied on the GPU during synthesis. The initial complex Fourier domain amplitudes $\tilde{h}_0(\mathbf{k})$, for example, are generated one time on the CPU and loaded into a static floating-point texture map using the expression in Equation 2. A separate floating-point texture storing the dispersion relation $\omega(k)$ is also loaded once at initialization time. All floating-point textures are stored at 32 bits per channel. As illustrated in the block diagram in Figure 2, the $\tilde{h}_0(\mathbf{k})$ and $\omega(k)$ texture maps can be processed using graphics hardware by rendering into another floating-point texture map using a 1:1 texel to pixel mapping as is typical in GPU-based image processing. The result of this step is a texture containing the Fourier domain amplitudes of our wave field at time t, computed by using a shader which evaluates the expression in Equation 4. The resulting texture is then transformed to the spatial domain using an IFFT, resulting in a realistic ocean water height field $h(\mathbf{x}, t)$.

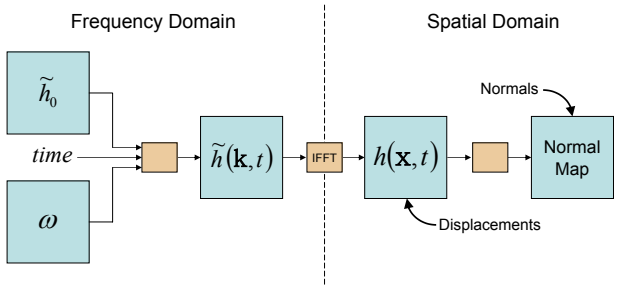


Figure 2: Ocean water synthesis process. Blue blocks represent texture maps and orange blocks represent kernels implemented with floating point pixel shaders.

Once we have the water height field $h(\mathbf{x}, t)$ in the spatial domain,

we use it to generate a world space normal map for shading. A simple cross product of x and y Sobel filters is used to compute the normal map in the spatial domain. Tessendorf describes a higher quality method for generating a normal map via an additional IFFT, but we have chosen to filter the spatial domain height map since it was straightforward to implement and because it fits naturally with the waveform compositing technique which will be described in Section 3.4.

In addition to computing a normal map for shading, we also use the spatial domain height map to vertically displace an underlying polygonal mesh. This is achieved by means of an experimental API extension which enables the hardware to interpret the output of the rasterizer as vertices which can be read in as an additional vertex stream, similar to techniques described in [Losasso02] and [Kipfer04]. An example of a highly agitated synthesized ocean surface is shown in Figure 3 with and without wireframe.

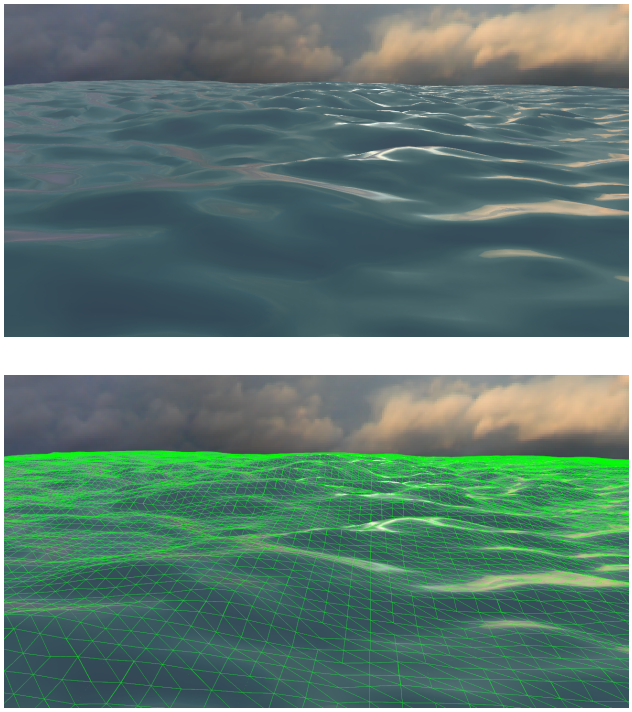


Figure 3: Real-time synthesized ocean water and corresponding wireframe

Since the synthesized height map pixels have a 1:1 correspondence to vertices in our tileable ocean water mesh, the height map does not create any sampling issues and does not need to be mip-mapped or filtered in any way. It is desirable to generate a mip-map for the resulting normal map in order to reduce aliasing, however. We use the GPU to downsample our computed normal map to fill out its mip-map each frame.

3.3 Multi-band Synthesis

In order to improve rendering speed or simulate depth effects, it is possible to synthesize multiple overlapping frequency bands of the ocean water height signal. As indicated in the block diagram in Figure 4, it is possible to synthesize two water surface height maps: a *broadband map* and a *low band map*.

The low band map is a low resolution texture which contains low frequency waveforms. This map is used to displace the water surface geometry. The broadband map is a higher resolution texture which contains the same low frequency information as the

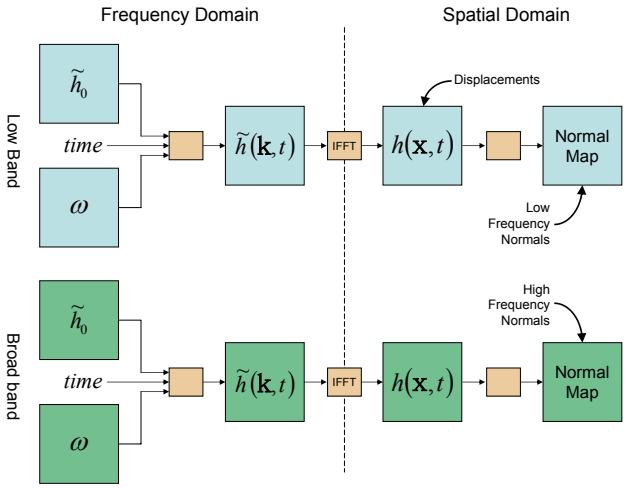


Figure 4: Dual band ocean water synthesis process. Blue and green blocks represent texture maps and orange blocks represent kernels implemented with floating point pixel shaders.

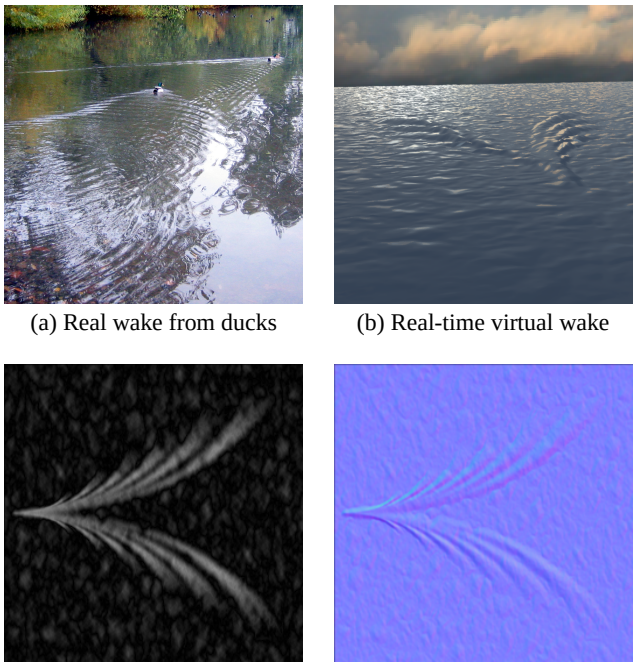
low band map plus higher frequencies. This height map is filtered to produce a normal map for shading, giving the appearance of a highly detailed ocean surface while maintaining a reasonable polygon count [Jensen01]. Figures 1a and 1b show the low band and broadband signals under the same viewing and lighting conditions. In these two images, the wind speed and hence the wave amplitude are low in order to simulate calm water. This same wind speed and viewpoint will be used in the shallow water rendering technique discussed in section 5 and shown in Figure 1c.

If an appropriate down-sampling filter were used, it would be possible to perform only the high-frequency IFFT and compute the low-frequency map in the spatial domain via downsampling. We have left this as a future optimization, however, and have chosen to perform two IFFTs each frame for our wave synthesis. In Section 5, we will describe a technique which blends between the broadband and low band maps to simulate depth effects and damping due to plant matter on the surface of the water. In the following section, we will discuss compositing of arbitrary waveforms with our synthesized height fields.

3.4 Compositing other waveforms

In the real world, water surface waves undergo constructive and destructive interference as shown in the image of wakes caused by real ducks on a pond in Figure 5a. The wakes from the multiple ducks in the pond interfere with each other and with other waveforms such as the low amplitude wind waves. In the graphics community, it has been demonstrated that it is possible to composite arbitrary waveforms such as wakes with real-time synthesized ocean waves [Hinsinger02],[Loviscach03]. In the offline domain, simply compositing simulated and authored height map animations was shown to be an intuitive method for designing controllable waves and ripples in small scale scenarios such as the mud jacuzzi scene in *Shrek 2* [Kofsky04]. We have implemented a similar scheme, which allows us to composite pre-authored height maps with our synthesized low band and broadband maps in the spatial domain prior to computation of the normal map. A typical synthesized height map has been composited with a pre-authored wake map as shown in Figure 5c. The wind speed is set fairly low in this case, so that the wake map's contribution to the resulting height field is readily apparent. This composite height field is filtered to compute the normal map shown in Figure 5d. Together, the displacement and

normal maps are used in the rendering of virtual wake as shown in Figure 5b.



(a) Real wake from ducks (b) Real-time virtual wake

(c) Height map with wake (d) Normal map of c

Figure 5: Compositing wake with synthesized water

Now that we have described a number of ways to generate and animate ocean water height maps—including multiple frequency bands and optional compositing of arbitrary waveforms—we will briefly describe a shading algorithm which can be used to realistically visualize the results in real time.

4 Rendering

While we have chosen to focus on GPU-based ocean water synthesis in this paper, it is certainly possible to use arbitrarily complex shading on the resulting mesh such as the algorithms described in [Nishita94] or [Premože00]. For visualizing our GPU-synthesized ocean waves, we assume we are above the water surface and render it using a fairly typical shading algorithm which includes reflective, refractive, Fresnel and fog terms:

$$\mathrm{fog}(s * (1 - r) + r * (env + sun)) \quad (5)$$

where s is the amount of light scattered from within the water, env is a reflective term from a cubic environment map representing the sky, sun is an optional Phong highlight corresponding to the sun contribution, r is a Fresnel term and the function $\mathrm{fog}()$ applies a distance based fog color.

Examples of real-time renderings using this shading model are illustrated in Figures 1, 3 and 5b.

This particular shading model does assume, however, that there are no local reflections or refractions due to a boat on the water or a shallow bottom visible beneath the water surface. In order to use this technique in a shoreline or pond setting, it would be straightforward to implement such local reflections and refractions using reflection and refraction mapping techniques as in [Vlachos02].

Since we can easily handle local reflections and refractions and because we would like to render small bodies of water in games, it is desirable to use this wave synthesis technique for shallow water. The next section will discuss the issue of applying our synthesis technique to shallow water as found in ponds or shorelines.

5 Shallow Water

Throughout this paper, we have specifically made simplifying assumptions by ignoring depth and hence shore interactions. As such, our model is technically applicable primarily to deep water rendering. Nevertheless, for the purposes of plausible wave rendering in interactive applications such as games, we expect that it will be straightforward to apply precomputed shore refraction to our waveforms to produce the basis of plausible waves approaching shore [Ts'o87] [Gamito02]. This could be done by simply warping the polygonal mesh representing the water surface in x and y (not height) such that it conforms to the contours of the shoreline.

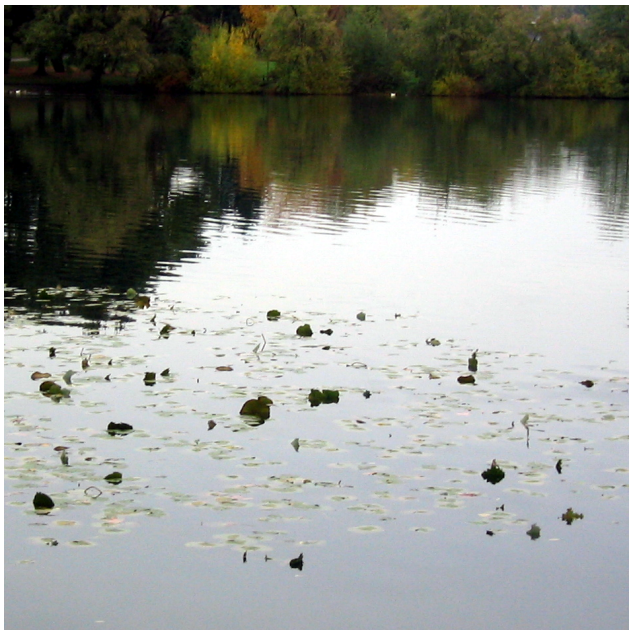
Even without addressing the wave refraction that one would expect to see on a rough coastline, we can simulate the appearance of the interaction of our synthesized waves with shallow regions or with plant matter found on the surface of ponds by exploiting our multi-band synthesis process. We can do this by using the low band map to displace the water surface geometry while blending between the low band and broadband normal maps as a means of filtering or damping out the high frequencies. In the real world, lily pads near the edge of a pond act as a filter which damps out the high frequency wind waves approaching shore as shown in Figure 6a. We can approximate this effect in our virtual pond, as shown in Figure 1c, by blending between the broadband and low band maps in our pixel shader according to a mask like the one shown in Figure 6c. In Figures 6b and 6c, we assume that the pond depth decreases from the upper left to the lower right. Figure 6c is a scalar factor which is used to transition between the broadband map and low band map so that high frequencies are gradually damped out by the lily pads, allowing only the low frequencies to pass through to the shore.

Shallow regions in the pond and variations in the wind above the pond can cause other regions of the water surface to undergo similar damping. This can be approximated in the same way, by damping out the high frequencies with an appropriate pre-authored texture map.

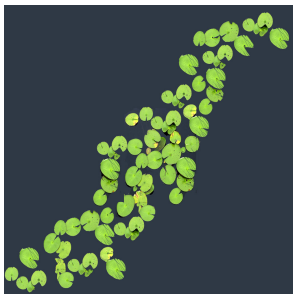
6 Performance

We have implemented this algorithm on a 2GHz Intel Pentium 4 CPU with 512MB of system RAM and an ATI RADEON X800 with 256 MB of video memory. For the dual-band synthesis, we have used a 64×64 low band grid and a 256×256 broadband grid. The GPU-based IFFT is a hand-tuned implementation based upon the Cooley and Tukey "Decimation in Time" algorithm [Cooley65][Mitchell03].

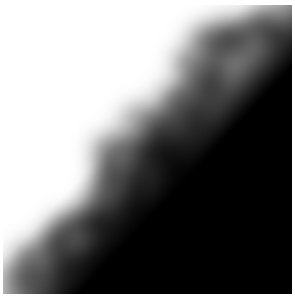
The overall performance is dependent upon the performance of both synthesis and scene rendering steps. On the ATI RADEON X800, with the scene rendering turned off, we are able to perform Inverse Fast Fourier Transforms for both the 64×64 and 256×256 floating-point images at 170 Hz. Turning on the GPU-based Fourier spectrum synthesis, followed by the IFFT and normal map filtering steps, we drop to 145 Hz. With wake compositing, this falls to 140 Hz. With scene rendering also turned on, we were able to achieve 65 Hz for typical viewpoints when rendering a 3×3 grid of instances of our tileable 64×64 water mesh (over 70,000 polygons) at a resolution of 768×768 pixels using the shading algorithm described in section 4. Naturally, one of the appealing properties of this approach is that the synthesis step has a fixed cost which is independent of the number of water surface tiles used in scene rendering. To further increase performance, one could imagine spreading the synthesis across multiple frames or only performing synthesis every n frames, interpolating between synthesized height fields for in-between frames.



(a) Real pond with damping from lily pads



(b) Lily Pad Texture



(c) High frequency damping map

Figure 6: Filtering of high frequencies by lily pads

As a means of verifying the correctness of our GPU-based ocean water synthesis and to provide context for performance analysis, we have also implemented this algorithm on the CPU using the FFTW library [Frigo98]. We compiled the FFTW library to use single-precision floating-point operations with version 8.1 of the Intel C++ compiler as was done on the BrookGPU project [Buck04]. Without doing the Fourier spectrum synthesis or normal filtering steps, we can compute the IFFT on 64×64 and 256×256 images on the CPU using the FFTW library at 165 Hz. This is consistent with other recent findings which show that current GPUs and CPUs perform roughly equivalently on the FFT [Buck04]. To come close to matching the GPU performance for the overall algorithm, however, we would also have to write highly optimized SSE code for the Fourier spectrum synthesis, wake compositing, normal map filtering and normal map mip-mapping steps. Even then, the data would be on the wrong side of the bus for the GPU to use it for scene rendering. We believe that these results make a strong case for performing ocean water synthesis on the GPU since the raw performance is compelling and because the resulting data is already in the right format and memory pool to be used by rendering algorithms

According to Tessendorf, the offline rendering community uses repeatable ocean water patches measuring 10 meters to 2 kilometers on a side, with the simulation resolution as high as 2048×2048 sample points, though significantly lower resolution simulations are often used. We believe that the performance of our algorithm is more than adequate to be considered seriously by the gaming community and we expect to see it adopted in future game titles.

7 Summary and Future Work

We have presented a Fourier domain approach to synthesizing and rendering deep-water ocean waves entirely on the graphics processor. Our technique is capable of generating and animating a Fourier domain spectrum of ocean water, which is then transformed to the spatial domain to obtain a realistic height map of ocean water. Low frequency height maps are used to displace geometry while broad spectrum waveforms are used to generate a normal map for shading. The model also allows for compositing of other waveforms caused by objects interacting with the water. We have also described a straightforward but realistic rendering equation which contains reflective, refractive, Fresnel and fog terms. We have also illustrated how a multi-band approach, which was initially conceived as a performance optimization, can be exploited to simulate shallow water effects found along shorelines or in small bodies of water. We concluded by demonstrating the performance of our algorithm on modern consumer graphics hardware.

We have concentrated our efforts on the synthesis of a repeatable tile of ocean water, which is inherently decoupled from level of detail (LOD) techniques which would be necessary when using this technique in a commercial game. We expect that straightforward LOD schemes like a quad-tree as applied in [Lanza04] or a set of nested grids as described in [Losasso04] will work well and we intend to explore this area.

An interesting related phenomenon which could be added to our algorithm would be the generation of repeatable caustic textures as described by Stam [Stam96][Stam]. This is interesting for games because spatially repeatable textures for refractive and reflective caustics could be generated on demand and hence have a small memory footprint. The fact that the caustics are correlated with the GPU-synthesized water surface may also be a desirable effect, depending on the specifics of the scene being rendered.

A number of papers using graphics hardware to perform fluid simulation or solve 2D wave equations have been published in the last few years [Harris03][Krüger03]. These simulations can be used to compute height maps which can be composited with our synthesized height maps in the same way that we have already demonstrated with precomputed wakes. We expect that this integration can be used to portray plausible real-time animations of dynamic ripples, eddies and general turbulence in our synthesized water.

Another extension which has been discussed in the CPU-based papers published earlier is post-processing of the synthesized height maps to increase choppiness and to spawn foam and spray. Specifically, Tessendorf suggests methods for processing the height field to cause the wave forms to resemble the superposition of trochoids rather than the sinusoids which are produced by the IFFT. The model presented here can easily be extended in this way.

Finally, it would be interesting to migrate the entire synthesis step to a stream processing model like Brook as a means of comparison with our hand-tuned model and to illustrate an additional application domain of the generic Brook system [Buck04].

8 Acknowledgements

Thanks to Evan Hart of ATI Research for help with the GPU-based FFT implementation.

References

[Buck04] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston and Pat Hanrahan, ”Brook

for GPUs," SIGGRAPH 2004.

[Cooley65] James W. Cooley and John W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series." Math. Comput. 19, 297-301, 1965.

[Fournier86] Alain Fournier and William T. Reeves, "A Simple Model of Ocean Waves", Computer Graphics, Vol. 20, No. 4, 1986, p 75-84.

[Frigo98] Matteo Frigo and Steven G. Johnson, "FFTW: An Adaptive Software Architecture for the FFT," *Proc. ICASSP* 3, 1381 (1998).

[Gamito02] Manuel Gamito and F. Kenton Musgrave, "An Accurate Model of Wave Refraction over Shallow Water," *Computers & Graphics* 26(2): 291-307, 2002.

[Harris03] Mark Harris, "Real-time cloud simulation and rendering," *Ph.D. dissertation*. University of North Carolina at Chapel Hill, 2003.

[Hinsinger02] Damien Hinsinger, Fabrice Neyret and Marie-Paule Cani, "Interactive Animation of Ocean Waves," Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2002.

[Isidoro02] John Isidoro, Alex Vlachos and Chris Brennan, "Rendering Ocean Water," *ShaderX: Vertex and Pixel Shader Tips and Tricks*, Wordware, 2002.

[Jensen01] Lasse Staff Jensen and Robert Goliáš, "Deep-Water Animation and Rendering," *Game Developers Conference Europe*, 2001.

[Kipfer04] Peter Kipfer, Mark Segal and Rüdiger Westermann, "UberFlow: A GPU-Based Particle Engine," *Proceedings Eurographics Conference on Graphics Hardware*, 2004.

[Kofsky04] Lewis Kofsky, "Image Based Fluids," SIGGRAPH Technical Sketch 2004.

[Krüger03] Jens Krüger and Rüdiger Westermann, "Linear Algebra Operators for GPU Implementation of Numerical Algorithms," SIGGRAPH 2003.

[Lanza04] Stefano Lanza, "Animation and Display of Water," in *ShaderX^3: Advanced Rendering with DirectX and OpenGL*, Wolfgang Engel editor, Charles River Media, 2004.

[Losasso02] Frank Losasso, Hugues Hoppe, Scott Schaefer and Joe Warren, "Smooth Geometry Images," *Eurographics Symposium on Graphics Processing* 2003, pages 138-145

[Losasso04] Frank Losasso and Hugues Hoppe, "Geometry Clipmaps Terrain Rendering Using Nested Regular Grids," SIGGRAPH 2004.

[Loviscach03] Jörn Loviscach, "Complex Water Effects at Interactive Frame Rates," Journal of WSCG 11, pp. 298305 (2003),

[Mastin87] Gary A. Mastin, Peter A. Watterger, and John F. Mareda, "Fourier Synthesis of Ocean Scenes," *IEEE Computer Graphics and Applications*, March 1987, p. 16-23.

[Max81] Nelson L. Max, "Vectorized procedural models for natural terrain: Waves and islands in the sunset," *Proceedings of the 8th annual conference on Computer graphics and interactive techniques*, p.317-324, August 03-07, 1981, Dallas.

[Mitchell03] Jason L. Mitchell, Marwan Y. Ansari and Evan Hart, "Advanced Image Processing with DirectX 9 Pixel Shaders" in *ShaderX^2 - Shader Tips and Tricks*, Wolfgang Engel editor, Wordware, Sept. 2003.

[Moreland03] Kenneth Moreland and Edward Angel, "The FFT on a GPU," SIGGRAPH/Eurographics Workshop on Graphics Hardware 2003 Proceedings, pp. 112-119, July 2003.

[Nishita94] Tomoyuki Nishita and Eihachiro Nakamae, "Method of Displaying Optical Effects within Water using Accumulation Buffer," SIGGRAPH 1994.

[Peachey86] Darwyn Peachey, "Modeling Waves and Surf," Computer Graphics (SIGGRAPH '86 Proceedings), volume 20, p. 65-74, Aug. 1986

[Perlin85] Ken Perlin, "An Image Synthesizer," SIGGRAPH 1985.

[Premože00] Simon Premože and Michael Ashikhmin, "Rendering Natural Waters," Eighth Pacific Conference on Computer Graphics and Applications, October 2000.

[Schneider01] Jens Schneider and Rüdiger Westermann, "Towards Real-Time Visual Simulation of Water Surfaces," Vision, Modeling and Visualization 2001.

[Stam96] Jos Stam, "Random Caustics: Natural Textures and Wave Theory Revisited" , Technical Sketch SIGGRAPH'96. In ACM SIGGRAPH Visual Proceedings, 1996, p. 151.

[Stam] Jos Stam, "Periodic Caustic Textures," online article www.dgp.toronto.edu/people/stam/reality/Research/PeriodicCaustics/

[Tessendorf99] Jerry Tessendorf, "Simulating Ocean Water," *Simulating Nature: Realistic and Interactive Techniques Course Notes*, SIGGRAPH 1999.

[Thon00] Sebastien Thon, Jean-Michel Dischler and Djamchid Ghazanfarpour "Ocean Waves Synthesis Using a Spectrum-Based Turbulence Function," *Proceedings of the International Conference on Computer Graphics*, 2000.

[Ts'o87] Pauline Y. Ts'o and Brian Barsky, "Modeling and Rendering Waves: Wave-Tracing Using Beta-Splines and Reflective and Refractive Texture Mapping," SIGGRAPH 1987. pp. 191-214.

[Vlachos02] Alex Vlachos, John Isidoro and Christopher Oat, "Rippling Reflective and Refractive Water," *ShaderX: Vertex and Pixel Shader Tips and Tricks*, Wordware, 2002.