An Open Programming Architecture for Modeling Ocean Waves

Lawrence M. Lachman
Senior Software Engineer
MultiGen-Paradigm
Richardson, Texas

ABSTRACT

Typical ocean simulations provide for wave modeling that is extraordinarily restrictive. The closed solution models ocean waves, with frequencies obtained from a particular energy spectrum. This energy spectrum and the underlying wave model are hardwired into the software such that users who wish to use a different energy spectrum, or a different wave formula and methodology are unable to do so.

Sophisticated computer programmers in this field (physicists, oceanographers, and naval architects) use their own wave models in order to compute the hydrodynamic motion and frequency response of their ships. However, they have no way to plug that same wave model into existing maritime ocean simulators. Thus, the ocean computed and rendered by these image generators has no bearing on the computed sway, surge, heave, yaw, pitch, or roll of the vessels, and if the vessel's position and orientation happens to look reasonable within the context of the rendered ocean (with respect to the waves surrounding the vessel), then that is merely a coincidence.

Our creative solution introduces new algorithms for rendering ocean waves in one unified manner, while simulation algorithms that are not native to any software system perform the generation and animation of the waves, modify the character of the ocean surface, and manage the underlying physics.

Presented at the IMAGE 2007 Conference
Scottsdale, Arizona July 2007

INTRODUCTION

There are many models for ocean behavior, each having multiple inherent assumptions and initial conditions, such that no single approach can be used to generate accurate, realistic results in varying environmental conditions. This is inherent in the nature of the ocean.

Ocean water simulation is a computationally expensive process and a difficult problem for applications that require visually plausible three-dimensional effects at high frame rates.

Our goals in designing an open architecture for ocean water simulation were to create a flexible and scalable framework (making it truly useful for a multitude of wave models), provide a high degree of visual fidelity regardless of how the waves are defined, and be fast enough to achieve realtime frame rates regardless of the complexity of the wave model.

The focus of this paper is the creation and rendering of wave geometry (and associated parameters); and not the rendering of the water surface, such as the color and clarity of the water volume or the reflection, refraction, and propagation of light.

We begin by introducing the topic of wave models. We will then review previous work in the field of ocean water synthesis. The framework of the open architecture is then detailed, followed by four empirical case studies that proved its viability. We conclude with a discussion of the latest work done in GPU-based wave simulation and its applicability to the open architecture.

WAVE MODELS

A wave model is a general term for numerical models designed to simulate the generation, propagation, shoaling, interaction, refraction, reflection, etc. of ocean waves. These are used to predict wave behavior for complicated wave fields and bathymetry.

Any wave system can be described as an array of wave groups that results from interference between several periodic, progressive wave trains that have generally different heights and periods, but which are all traveling in nearly the same direction.

Wave Trains

The general equation for a sinusoidal wave is:

$$\zeta(X) = A \cos(kX - \omega t) \qquad (1)$$

where

- $\zeta(X)$ is the wave elevation at geographic location X.
- A is the amplitude.
- k is the wavenumber.
- X is the geographic location ($X = x \cos\theta + y \sin\theta$; θ is the wave direction with respect to the coordinate system), and kX represents the space variation.
- ω is the angular frequency in radians per second.
- t is the elapsed time since the beginning of the simulation, and ωt is the time variation.

A wave train is a succession of sinusoidal waves arising from the same source, having the same characteristics and propagating along the same path. A long-crested wave is formed when energy comes from one direction. The crests lie in parallel rows and the height does not vary along a given crest.

Traditionally, ocean waves are simulated by superposing many periodic wave trains of different frequencies and amplitudes, advancing in different directions, (with the wave energy) spread around a predominant direction. Such waves are called short-crested. The wave crests are not exactly parallel, are of finite length, and vary in height (along the crest).

As the simplest example, figure 1 illustrates two identical wave trains having slightly different periods (and, hence, different velocities) propagating in a body of water. At any point, the instantaneous surface elevation will be the arithmetic sum of the amplitudes of the two component wave trains, shown by the thick curve in figure 1.

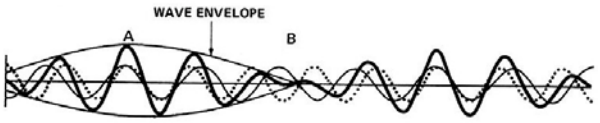

Figure 1. Wave groups are formed by interference between two or more wave trains of different periods moving in the same direction. Here, the component trains (light continuous and dotted curves) add arithmetically to produce a composite train of variable amplitude (thick curve). The bounding envelope defines a wave group.

At point A, two component crests happen to coincide, and the wave height is doubled; at point B they are opposed and cancel one another. The effect of interference is to divide the mixed wave train into groups of waves – indicated by the bounding wave envelope – that are separated by nodes where the wave height vanishes.

Wave Spectra

A wave spectrum is a statistical representation of the distribution of wave energy as a function of the wave frequency.

The surface of the sea contains waves of many frequencies and amplitudes that add together to produce the spectrum of a developed sea. The waves in a particular sea state are identified by their wave spectrum, $S(\omega)$, and wave spectra are constructed based upon empirical studies that describe oceanic conditions (i.e., parameterized formulae are developed by spectral analysis of measured wave data).

Directional spectra are customarily expressed in the form

$$S(\omega,\theta) = S(\omega) G(\theta, \omega) \qquad (2)$$

where $S(\omega)$ is a standard long-crested wave spectrum and G is a directional spreading function. Directional spreading functions describe directional distribution of wave energy.

Spectral Methods of Wave Generation Prediction

Ocean waves are, for the most part, produced by the wind (seaquakes and other disturbances of the water also produce waves). When the wind blows over a calm, almost smooth sea, the turbulence in the wind

produces random pressure fluctuations at the sea surface.

In 1952 researchers began developing mathematical representations of wind-generated wave phenomena. Their efforts laid the groundwork for the definition of a wave spectral density function that could predict wave amplitude based upon certain wave components. These spectral methods turned out to be an excellent way to visualize wind effects, and served to reinforce the validity of spectrum analysis.

In order to obtain the spectrum of ocean waves at the downwind side of an area, various idealized, analytical and empirical spectra are used in oceanography. The most common spectra are: *Bretschneider, Pierson-Moskowitz, JONSWAP, and Phillips*. Depending upon the model, the underlying formula describes a fully developed sea as determined by one, two, or three of the following parameters: wind speed, significant wave height, and modal period.

The *Bretschneider* (1956) spectrum was developed for the North Atlantic, unidirectional seas, with infinite depth, no swell and unlimited fetch. This approach, while valid, was improved upon by the *Pierson-Moskowitz* (1964) spectrum.

The *Pierson-Moskowitz* spectrum was also developed for the North Atlantic but incorporates two significant assumptions. This spectrum assumes that waves can be represented by Gaussian random processes. It also assumes a fully developed sea in which the spectrum no longer grows given a constant wind velocity. The *Pierson-Moskowitz* spectrum serves as the basis for standardization of the prediction of seafaring vessels and offshore structures in the open sea.

Developed during the Joint North Sea Wave Observation Project, the *JONSWAP* (1973) spectrum is similar to the *Pierson-Moskowitz* spectrum except that waves continue to grow with distance or time (the wave spectrum is never fully developed), and the peak in the spectrum is more pronounced

The *Phillips* (1957) spectrum is a statistical model where wind represents a random distribution of pressure fluctuations as idealized circular gusts. It relies on the turbulent pressure fluctuations to provide random forces acting on the water surface that ensures wind-to-wave energy transfer. Within this model the wave energy and amplitude increase linearly with time.

A detailed discussion of the various formulae is beyond the scope of this paper. However the interested reader is directed to bibliographic entries [5] and [12].

WAVE GENERATION AND ANIMATION

There is a substantial body of literature on ocean water simulation and animation, in the context of oceanography and computer graphics.

Oceanographic numerical models can be divided into two classes: mechanistic models and simulation models.

Mechanistic models are simplified models used for studying processes such as describing planetary waves, the interaction of the flow with seafloor features, or the response of the upper ocean to the wind.

Simulation models are used for calculating realistic circulation of oceanic regions. These models are often very complex.

Ocean water synthesis for computer graphics can also be broken into two classes: spatial domain and Fourier domain approaches.

Spatial Domain

Physics-based wave models work reasonably well when modeling infinitely deep water, however, they are computationally expensive. This is especially true when attempting to model large bodies of water, because in the spatial domain the numerical model must be evaluated at each vertex, as it is dependent upon time and space.

The complexity of the wave model depends on the purpose of the simulation. The continuum ranges from simulating infinitely deep water (where bathymetry and the history of the waves are not accounted for) to the most complex case of simulating the propagation of waves from deep water into a shallow water region. In the latter case the wave model must account for reflection and refraction of each wave train component.

The representation of spatial wave geometry may be considered at two levels. At a high level, the overall appearance of the waves is affected by the energy spectrum (heights, modal period, wind speed, etc.) and the spreading function. At a low level, the detailed appearance of the wave, particularly in the region of the wave crest, is affected by the shape of the wave profile, sine wave, trochoid, distorted sine wave, etc.

Several graphics researchers have studied the large scale motion of water in waves in the spatial domain. The seminal works by Fournier and Reeves [3] and Peachy [15] relied on a mix of Gerstner and Biesel swell models.

Gerstner waves model trochoidal-shaped waves (figure 2). While Gerstner waves were developed long before computer graphics to model ocean water on a physical basis, oceanographic literature has tended to downplay them as a realistic model of the ocean.

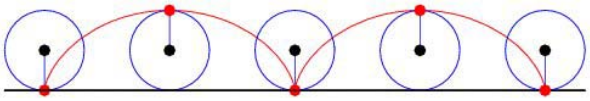

Figure 2. A trochoid is a curve described by a fixed point as a circle rolls along a straight line. A trochoid has parametric equations:

$$x = a\theta - b\sin(\theta), \quad y = a - b\cos(\theta)$$

where θ is the variable angle through which the circle rolls.

Peachy generated a height field by computing the superposition of several long-crested waveforms. Instead of using trochoids, Peachy blended a quadratic function with the underlying sinusoids to give a more realistic cycloidal choppy appearance.

Thon [17] computed the height field 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.

Max [11] generated a height field composed of the summation of low amplitude sinusoidal wave trains with added noise. Max chose the frequencies of these waveforms so that they would repeat at a common point in time so that their motion could be looped and reused throughout an animation.

While many other graphics researchers have contributed in this domain, Perlin's [14] noise synthesis approach and Ts'o and Barsky's [18] wave tracing approach are two others that are noteworthy.

Fourier Domain

Fourier domain approaches synthesize ocean water by utilizing measured spectral properties and known statistical models of ocean water.

This approach, first introduced to the computer graphics community by Mastin et al. [10], synthesizes and evolves a patch (mesh) of ocean waves from a Fast Fourier Transform (FFT) precept, using a regular grid with user-controllable size and resolution, and yields a periodic height field that can be tiled seamlessly over a larger domain. This approach simulates many different waves simultaneously, with visually pleasing results.

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

Tessendorf's "Simulating Ocean Water" [16] described a similar approach which has been applied to production rendering in films such as *Waterworld* and *Titanic,* as well as realtime, maritime simulation software [9]. Tessendorf starts in the frequency domain, using the *Phillips* spectrum, and then transforms a Fourier domain description of the ocean water to the spatial domain via the inverse FFT. Tessendorf also modified the synthesized height field to yield trochoidal waves (the IFFT, by definition, does not directly result in trochoidal shapes).

One drawback of this approach relative to more costly spatial domain approaches is that, at high altitudes, it is evident that the synthesized ocean water surface is comprised of repeating tiles. However, there are rendering techniques that can reduce and ameliorate the periodicity, and for many types of interactive applications, this repetition is not apparent to the user.

Fourier domain approaches do not account for the effects of bathymetry (i.e., a varying bottom profile is not directly solvable with FFTs). However, this approach can be used with other methods where the effects of bathymetry are used to generate the input parameters related to the wave characteristics [9].

THE OPEN ARCHITECTURE

Our goal in designing the open programming architecture was to create a flexible and scalable framework that provided an open and intuitive interface for incorporating any one of a multitude of wave models, regardless of their complexity. At the same time, we endeavored to decouple the complexity of the wave model from the methodology in which its results were rendered. This led to the design of a single, high fidelity rendering engine.

The work in ocean water synthesis (cited previously) has led to the development of a variety of realtime, maritime simulation software packages. Whether they are custom developed image generators or COTS (commercial, off-the-shelf) software, they all have had one thing in common: they provide for wave modeling that is extraordinarily restrictive. Their closed solution for modeling ocean waves is hardwired into the

software such that users who wish to use a different wave model and methodology are unable to do so. Furthermore, the rendering methodology is tightly coupled to the chosen domain (spatial or Fourier).

Our solution features a multi-layer architecture and consists of four main components: an ocean, a wave generator, a geometry grid, and a rendering engine.

In order for our solution to run at realtime frame rates, it updates the wave generator and constructs the ocean asynchronously from the frame loop, and it renders the ocean using range-based level of detail and adaptive triangle meshes.

The Ocean

Our discussion begins with the terminology used in the context of simulating ocean water.

- An ocean is a single level (of detail) rectangular grid of cells. The ocean is divided into columns and rows (the grid structure), which define the individual cells of the grid (figure 3a).
- A cell is a subdivision of the grid structure. Each cell has a unique column and row number (address). A cell is independent of the geometry inside of it; it simply identifies the boundaries and address of a location within the grid structure (figure 3b).
- A mesh is the geometry that occupies the cell of a grid structure. A mesh is a collection of uniformly spaced, discrete points arranged in a roughly rectangular pattern (figure 3c). The points of the mesh correspond to multi-component data elements (e.g., height (1D), position (2D), and normal (3D)). While the horizontal dimensions (extents) of a mesh are constant throughout an ocean grid, each mesh may have a different resolution (level of detail) that is a function of range from an observer (figure 3d).
- A mesh point is the smallest indivisible element of a mesh (figure 3e); while four mesh points comprise a mesh facet (figure 3f).

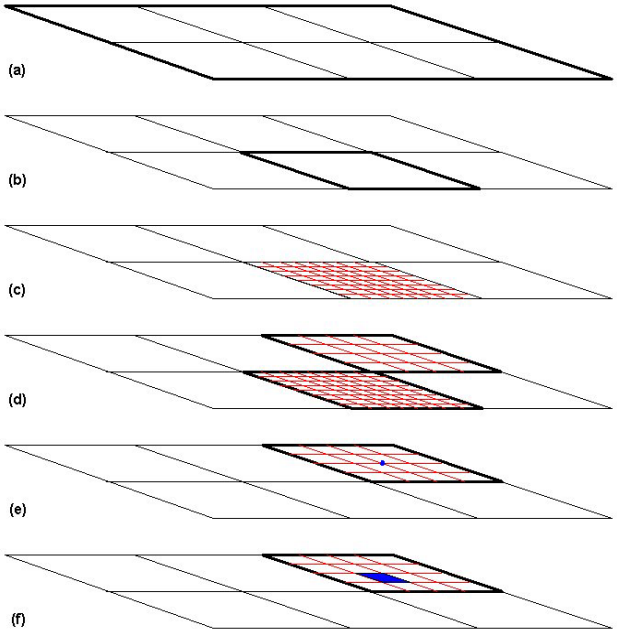

Figure 3. Topology of the ocean. (a) An ocean with a grid structure of 2x3 cells. (b) Cell (1, 0) of the grid structure representing a physical region. (c) A mesh with a resolution of 8x8, co-located in cell (1, 0). (d) Two meshes with different resolutions; cell (1, 0) with a mesh resolution of 8x8, and cell (2, 1) with a mesh resolution of 4x4. (e) A mesh point. (f) A mesh facet.

The ocean class features range based level of detail management, where ocean cells are generated with different levels of polygonal detail. The mesh density of all cells is determined dynamically, and the highest resolution is rendered relative to the proximity of the observer. The underlying geometry is unique per observer. This enables the cell's level of detail and optics (i.e., the reflection, refraction, and propagation of light) to be unique for each observer.

The Wave Generator

An ocean instance contains a wave generator. The wave generator acts as the interface to the various controls and data generators for basic, choppy, and shallow water waves.

The WaveGenerator class is an abstract base class that defines the interface for derived classes that perform the animation of ocean water. It is an implementation of the template method design pattern [4], as it:

- Defines the skeleton of the update and configuration of the wave generator, and defers the behavior that can vary to subclasses.

- Specifies the complete interface through which users may access and manipulate instances of derived concrete classes.

It provides a generic, comprehensive, and extensible set of abstractions that allow the definition and use of interoperable wave model components. It is the responsibility of the user-derived class to provide the content for many of these abstractions.

Once a derived instance has been created, a pointer to the instantiated object is passed around the software's infrastructure as a pointer to an object of type WaveGenerator with no compile-time dependencies whatsoever on its implementation.

Accordingly, derived classes can incorporate completely foreign wave simulation algorithms into the simulation at a low level, and have the resulting waves handled and rendered equivalently to other types of wave models that are native to the software.

The interface for the WaveGenerator class declares a set of pure virtual functions; functions that subclasses must define and implement. It is the derived class' responsibility to:

- Determine whether the wave generator produces a surface that can be treated as periodic when sampled in surface queries.

  A wave generator is considered periodic if the waves produced by it form a complete cycle at the interval that coincides with the physical dimensions of an ocean cell (i.e., surface queries at that interval produce identical results). A periodic surface enables certain rendering optimizations to be employed, which results in a substantial performance benefit.

- Answer the following raw surface property queries at a specified geographical location:

  - Height
  - Normal
  - Velocity

- Fill in the grid of geometry data for the underlying mesh of a specified ocean cell, at the requested level of detail.

- Update the wave generator. In the spatial domain, this is where the time-dependent portion of the wave calculations would be performed. In the Fourier domain, this is where the generation of the height map, normals, velocity calculations, etc. would be performed.

- Configure the wave generator. This functionality is invoked when the wave generator is configured for the first time and again after significant parameters have been modified.

- Swap the active and inactive wave data pointers. Our solution spawns an asynchronous thread that performs the update, paging calculations, and construction of each ocean. These are computationally intensive tasks that, depending on the complexity of each ocean, may take longer than a single frame to finish. Consequently, wave data is distinguished between active (rendered) and inactive (under construction) wave data.

- Set the bounding 2D rectangle that defines the ocean, and the region over which ocean queries produce usable results. Our solution simulates bounded as well as unbounded oceans.

The WaveGenerator class also declares a set of virtual functions and, for each one, provides a default implementation that derived classes can optionally override. The derived class can:

- Declare that it can produce shallow water waves.
- Answer depth queries at a specified geographical location.
- Examine the surface of the water of a specified ocean cell to determine and record where breaking waves occur.
- Freeze or thaw the water surface at a given state.
- Set the dominant wave direction (the angle that the waves flow towards). When modeling short-crested waves in the spatial domain, waves spread out around this direction, up to plus or minus a specified angle of incidence (usually 45 or 90 degrees).
- Set the modal period.
- Set the sea state.
- Set the factor that controls the choppiness of the waves. This control is for wave models that have the ability to form sharper peaks and broader, deeper troughs than basic, rounded waves.
- Generate a high resolution normal map. When modeling ocean waves in the Fourier domain, a high resolution normal map facilitates per-fragment lighting and per-fragment environment mapped bump mapping.

Finally, the WaveGenerator provides non-virtual functions for setting spectrum parameters, surface wind

speed and significant wave height, and a function for setting a peak threshold value for detecting whether a point on the ocean surface is near the peak of a wave.

The latter control is for derived wave generators that are capable of measuring and detecting when the peaks of waves have become very sharp, signaling that a wave should or could break. Once these peaks are identified, whitecaps can be rendered at those locations. Below this region no white cap or foam is rendered. Within this region white caps are rendered in a gradient fashion, reaching full strength at the wave's crest.

The Geometry Grid

A WaveGenerator instance contains one GeometryGrid, a container class that can hold single or multi-channel data in a rectangular grid suitable for periodic or non-periodic grids. The geometry grid data is in the form of VertexData, a structure containing various types of raw surface properties such as height, depth, normal, foam factor, and displacement, at each point along the grid. While derived wave generators can selectively choose which properties to compute, height is the only surface property that must be computed.

Velocity is intentionally not included in the geometry grid data. This is because velocity vector and group velocity calculations add substantially to the computational load and it is more practical to compute the specified velocity only where queries are made (versus at every vertex).

The resolution of the GeometryGrid is congruent with the highest attainable mesh resolution, and it is filled in during the construction of the underlying geometry of each ocean cell. The infrastructure subsequently samples the GeometryGrid in order to construct the underlying mesh of each ocean cell.

The rendering engine consumes this unified data structure. It is unified in the sense that all wave generators produce it in the same format regardless of the underlying wave model that the wave generator implements.

The Rendering Engine

Over the last two decades there has been extensive work done in the area of terrain visualization and level of detail management. Gross et al. [6] were among the first to propose a method for adaptive mesh tessellation at near interactive rates. Duchaineau et al. [1] proposed the seminal ROAM algorithm for interactive, view-dependent refinement of terrain.

Of the existing algorithms in the current literature that can interactively perform view-dependent, locally adaptive terrain meshing, all rely on a pre-defined multiresolution representation from which to build the adaptive triangle meshes each frame. Furthermore, there is a substantial body of literature that addresses the problem of laying out the terrain data on disk to achieve efficient performance.

Hinsinger et al. [7] were the first to develop a technique that adaptively tessellates a grid of points for interactive rendering of Gerstner waves in the spatial domain. This tessellation scheme subdivided the screen into a grid of quads, which are back-projected on the plane modeling the ocean surface at rest. That is, vertices are evenly spaced, in post-perspective camera space (the projected area on screen), not in world space which is the traditional way. This technique approached realtime frame rates with a very coarse resolution (rendered at 20 – 30 Hz), while a very fine resolution rendered at less than 1 Hz.

The same year that Hinsinger et al. proposed their technique, we were creating our technique for realtime rendering of self-adapting ocean meshes. While both approaches are performed on the CPU, our main contributions here are:

- Our adaptive scheme applies to the geometry, and is independent of the animation process.
- Our approach can incorporate both spatial and Fourier domain approaches.
- Our approach achieves realtime frame rates of 60 Hz for coarse resolutions, and 30 Hz for fine resolutions.
- Our approach can be utilized by multiple channels and multiple observers, and still run at realtime frame rates.

All were key requirements that drove the design of the open architecture.

Hinsinger et al.'s adaptive scheme was tightly coupled to the animation process, and their model could only incorporate spatial domain approaches.

The per-cell level of detail determination, wave model computations, and vertex list generation, occur in the context of an asynchronous thread that synchronizes the construction of its geometry with the main application thread. Ocean cells feature the ability to be view frustum culled. If the ocean cell is not completely or partially within the viewing frustum, then no overhead is incurred to construct it. This substantially decreases the construction time of the ocean.

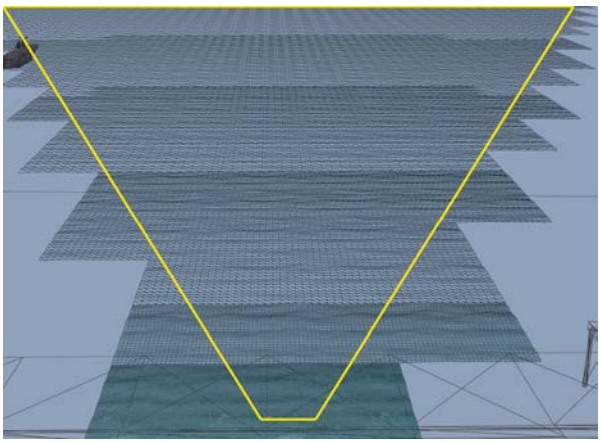

Figure 4. View frustum culling of the adaptive triangle meshes featuring range-based level of detail.

Each ocean cell is aware of the level of detail of its underlying geometry and that of its two neighbors (to the east and to the north). Thus, each ocean cell is rendered, potentially, in a different level of polygonal detail each frame. The resolution decreases the farther cells extend out. If an east and/or north neighbor cell exists, then the right side and/or top of the mesh are stitched correctly to correspond to the resolution of the corresponding neighbor (figure 5). Otherwise, the right and/or top of the mesh are rendered at the resolution of the cell being examined.

When an ocean cell pages into memory, it queries the east and north cell neighbors (if they exist) so it can then query their level of detail each frame, and notifies the west and south neighbors (if they exist) that it has just paged in.

Conversely, when an ocean cell pages out of memory, it resets its east and north neighbors to NULL, and notifies the west and south neighbors (if they exist) that it has paged out.

An ocean cell contains one ocean mesh per observer. This enables the cell's level of detail and optics to be unique for each observer. An ocean mesh contains two underlying geometry instances, as the geometry is double buffered, with one instance being rendered, while the other is constructed for the next animation sequence.

In the cull thread multiple channels access and share a pointer to the ocean geometry that corresponds to a common shared observer.

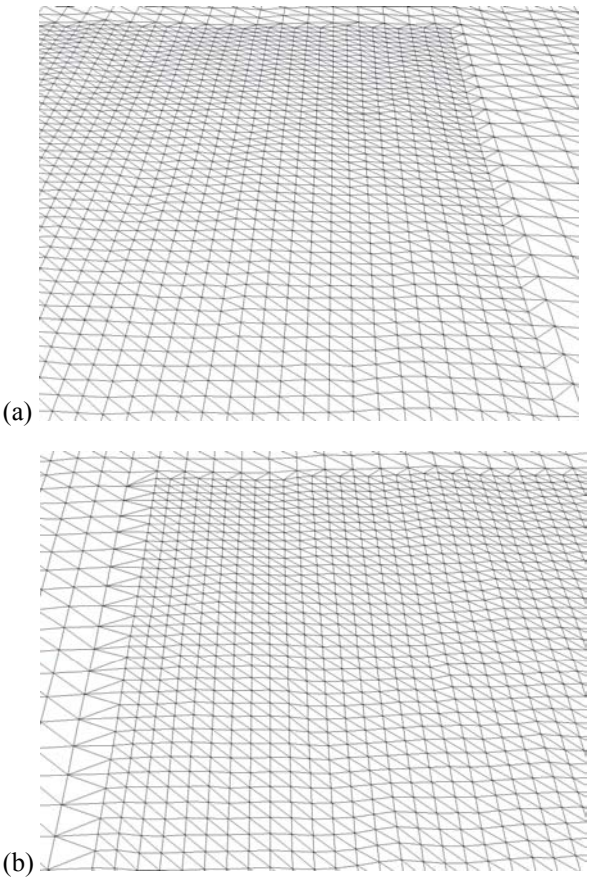

Figure 5. Variant polygonal stitching between an ocean cell and its east and north neighbors.
(a) Both neighbors have a lower mesh resolution.
(b) The ocean cell on the left is stitched to its east neighbor with a higher mesh resolution, while its neighbor to the north is of the same resolution. The ocean cell on the right is stitched to a northern neighbor with a lower resolution.

In order to support both the spatial and Fourier domains, the construction of the geometry requires that the resolution of each mesh be a power of two in both the x and y dimensions, and the horizontal dimensions (width and length) of all meshes are constant throughout an ocean grid.

Significant performance gains result from organizing triangles into strips. The mesh is rendered as (at most) four triangle strips. Almost the entire body of the mesh comprises the first strip. The remaining three strips are the right, top, and corner stitching, respectively.

The resolution of each mesh determines the minimum wavelength that can appear within the ocean cell (i.e., we do not compute or display waves which appear smaller than a mesh facet). As the range increases and the polygonal density becomes more coarse, this saves

much computational time, since we don't need to simulate the associated wave trains for this frame and location. User-configurable switch ranges allow for the tuning of the geometric complexity and the avoidance of popping artifacts (waves suddenly appearing or disappearing at certain distances).

An important consideration in properly setting the user-configurable switch ranges is that short wavelength components may contribute significantly to the local wave elevation (peak) when superposed with other components. Culling the short wavelengths may result in the significant reduction of the peak wave elevations which may be noticeable within a certain range.

View dependent, realistic optics and shading are natively provided by our solution's employment of hardware shader technology.

We apply an environment map on the ocean surface, thus reflecting the sun and the sky. Moreover, the rendered color of the surface is the product of the Fresnel reflectivity times the sky color. This is actually the correct physical lighting process, uncompromised by realtime rendering. To further increase the realism, we dynamically render local reflections on the ocean surface (i.e., ships, lights, buildings, etc.).

Vertex and fragment shaders are also used to render the white caps, foam, and backwash.

While a detailed discussion of the view dependent optics and shading are beyond the scope of this paper, what is pertinent here is that the wave geometry can only represent waves whose apparent length is some multiple of the grid size. For realtime rendering this means that waves shorter than *n* meters (where *n* is smaller than a mesh facet) cannot be rendered in geometry. However, the appearance and fidelity of the water surface is completely dependent on short waves. Our rendering engine determines how best to represent the effect that the environment has on these short waves and the resulting effects on the water surface appearance.

EMPIRICAL CASE STUDIES

Our research led to the successful development of several empirical case studies that also served as proof of concepts intended to prove the viability of, and test the open architecture. We present two case studies from each domain. In all four case studies, MyWaveGenerator is derived from the abstract base class, WaveGenerator.

Spatial Domain Case Study 1

Simulate wind generated waves by superposing N wave spectra over the most general form of the wave field $\zeta(x,y,t)$ corresponding to a given spectrum $S(\omega,\theta)$.

$$\zeta(x,y,t) = \int_{0}^{\infty}\int_{-\pi}^{\pi} \cos\{k(\omega)[x\cos\theta + y\sin\theta] - \omega t + \varepsilon(\omega,\theta)\}\sqrt{S(\omega,\theta)}\,d\omega\,d\theta$$

MyWaveGenerator implemented a discreet version of the general form of the wave field, which resulted in the computation of a finite number of directional wave trains for modeling of short-crested waves.

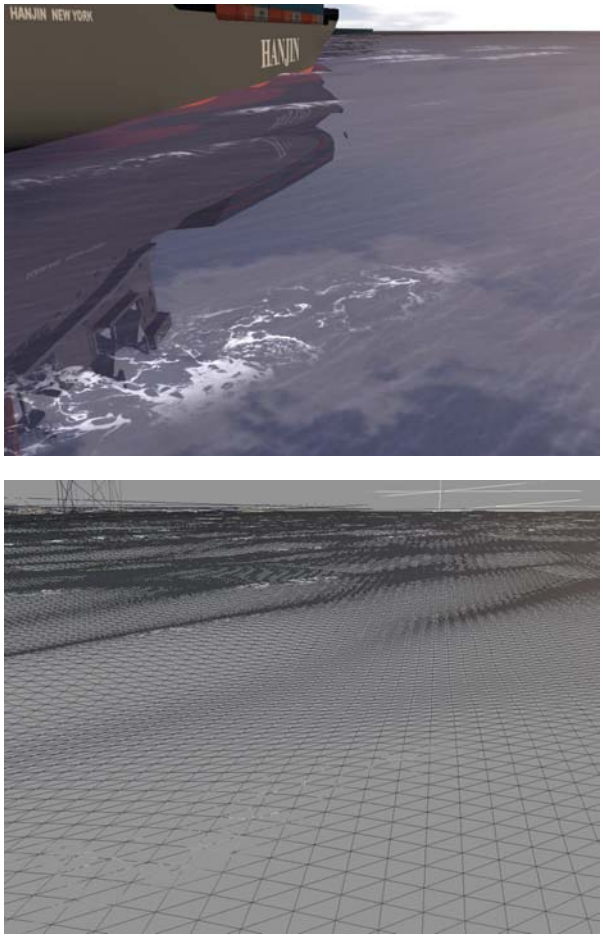

Figure 6. Realtime synthesized ocean water and corresponding wireframe in the spatial domain (case study 1).

It employed the *Bretschneider* spectrum, and simulated up to five directional wave trains (each within ±90 degrees of the dominant wave direction). For each of the wave trains, the user provided the number of frequencies per wave train, the direction of the wave

train, and the value of each component (phase angle and angular frequency).

MyWaveGenerator set the sea state according to the *Spectral Ocean Wave Model* [12], a northern hemisphere hindcast mathematical model that accounts for modal period, as well as wind speed and significant wave height, in its characterization of sea states.

Spatial Domain Case Study 2

Simulate wind generated waves with one wave train comprised of N directional components over the most general form of the wave field $\zeta(x,y,t)$ corresponding to a given spectrum $S(\omega,\theta)$.

MyWaveGenerator implemented a discreet version of the general form of the wave field, which resulted in the computation of one spectrum with multiple directional components for the modeling of short-crested waves.

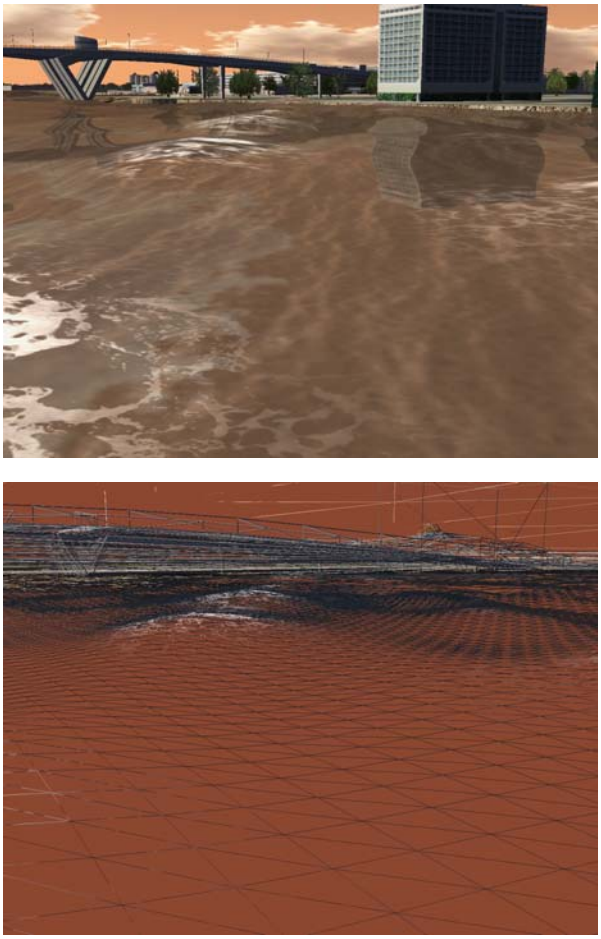

Figure 7. Realtime synthesized ocean water and corresponding wireframe in the spatial domain (case study 2).

It computed the height field by selecting 11 low amplitude sinusoids from the *Pierson-Moskowitz* spectrum, and directly calculated their superposition in the spatial domain.

The components of this short-crested wave spectrum were characterized by one predominant wave heading, while each component was required to be within ±45 degrees of the dominant wave direction.

MyWaveGenerator set the sea state according to the *Beaufort Wind Force Scale*, one of the first scales ever created (1805) to estimate wind speeds and the effects on the ocean.

Fourier Domain Case Study 1

Simulate short-crested, infinitely deep water ocean waves by synthesizing a height field using the Fourier domain approach.

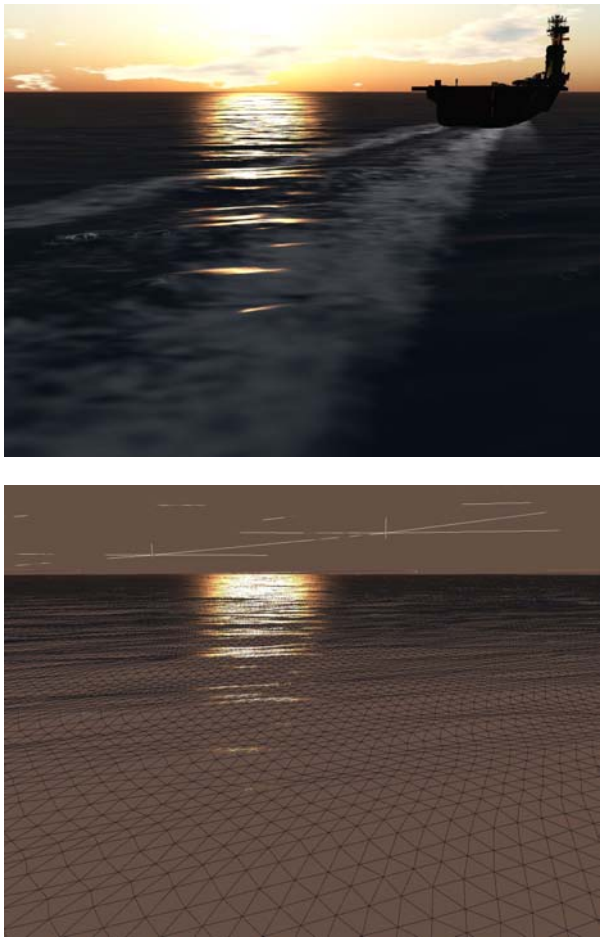

Figure 8. Realtime synthesized ocean water and corresponding wireframe in the Fourier domain (case study 1).

MyWaveGenerator employed an oceanographic statistics-based surface wave model for ocean waves. This model allows control of sea state, including distributions of direction, height, wavelength, and alignment with the wind. The algorithm is explained in detail in [16].

The height map was filtered to produce a high resolution normal map for shading, giving the appearance of a highly detailed ocean surface while maintaining a reasonable polygon count.

MyWaveGenerator set the sea state according to the *Beaufort Wind Force Scale*.

Fourier Domain Case Study 2

Simulate shallow water behavior. Visualize almost vertical wave fronts that result as a function of the bottom depth. Transition the water surface at the shoreline where the water tapers to zero height at zero depth, and transition seamlessly from shallow water to deep water.

MyWaveGenerator modeled the propagation and shoaling of waves in a shallow water area, as well as their somewhat short-crested character. The wave generator managed two regions: the littoral zone and the deep water region. It employed Fourier synthesis to generate its oceans, which enables meshes to be periodic and ocean cells to be tiled seamlessly over a larger domain.

This realtime solution exploited mesh periodicity in the deep water region by modeling one ocean cell and then tiling it horizontally and vertically throughout the entire region. In the littoral zone, because each zone could have a different slope, bathymetry was automatically generated for one ocean cell to correspond to each zone's slope. The accurately modeled shallow water surface of that cell was then tiled (horizontally) across the entire zone [9].

MyWaveGenerator set the sea state according to the *Pierson-Moskowitz* sea state table.

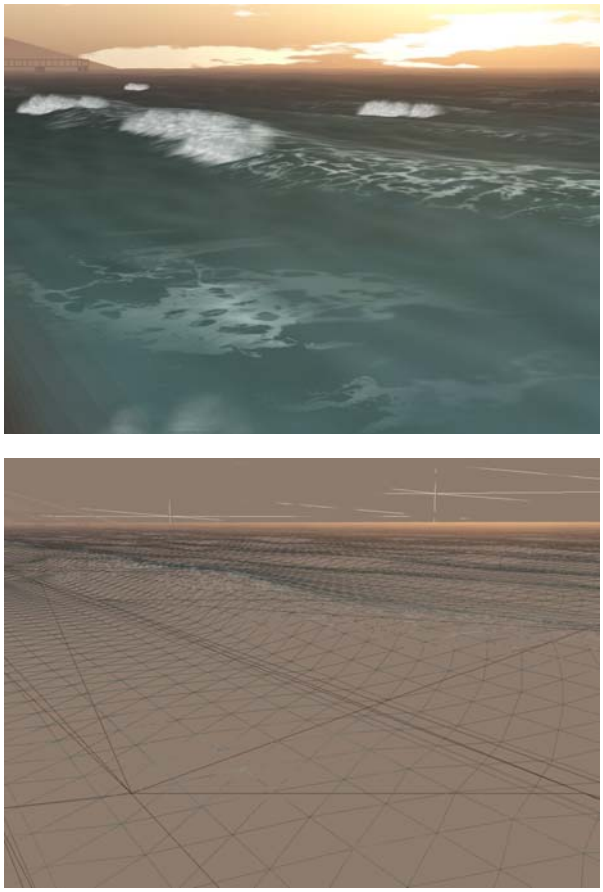

Figure 9. Realtime synthesized ocean water and corresponding wireframe in the Fourier domain (case study 2).

GPU-BASED WAVE SIMULATION

The advances in graphics hardware technology over the past several years have given rise to multiple approaches that have generated plausible realtime water surface animation entirely on the graphics processor.

Finch [2] and Woodard [20] employed Gerstner waves to simulate ocean waves. They computed geometry displacement as the superposition of four low-frequency sinusoids using hardware vertex shaders. Corresponding normals were also computed in the vertex shader. Because vertex data is no longer dynamic on the CPU side (the GPU is performing all of the wave simulation), Woodard employed vertex buffer objects to store the geometry for a mesh directly in graphics memory, eliminating the need to re-transmit the data across the graphics bus each frame.

Kryachko [8] used vertex texture displacement to combine two octaves of gradient noise to perturb a height field at interactive rates for realistic water

rendering. Kryachko used a static radial grid, centered at the camera position, to tessellate the water surface. However, the number of tessellated vertices that ended up within the view frustum was approximately 25 percent.

Finch, Woodard, and Kryachko synthesized ocean water in the spatial domain, which has the advantage that only visible regions and frequencies of the ocean surface are evaluated for each rendered frame. However, when modeling large bodies of water, it is still unclear whether physics-based wave models are readily adapted for the GPU, primarily because they require too many spatial domain terms for acceptable performance. Fourier domain synthesis does not explicitly compute the superposition of spatial domain sinusoids. As a result, the computational load on the GPU is less.

Mitchell [13] presented a GPU-based multi-band Fourier domain approach to synthesizing and rendering infinitely deep water ocean waves. However, this approach did not incorporate level of detail schemes.

Simulating ocean waves entirely on the graphics processor has several untoward consequences. Raw surface queries (height, normal, velocity, etc.) cannot be performed on the GPU; they can only be performed on the CPU. As a result, the wave model must be duplicated on the CPU, and for multi-threaded applications (i.e., those with a separate draw thread) this incurs a significant amount of complexity and overhead. Also, GPU-based approaches cannot account for multiple view-dependent metrics and still run at realtime frame rates.

CONCLUSION

We have presented a flexible and scalable framework that features a multi-layer open architecture for modeling ocean waves with simulation algorithms that are not native to any software system.

Our approach can incorporate both spatial and Fourier domain wave models, and achieves realtime frame rates for configurations utilizing multiple channels and multiple observers.

The single rendering engine employs vertex and fragment shaders to render view dependent, realistic optics and shading; and manages the paging of ocean cell geometry, culling ocean cells to the view frustum, the level of detail of the geometry, and triangle stripping. These algorithms produce optimal triangle meshes of varying polygonal density, where the density is view dependent, independent of the animation process, and where all logic:

- is performed at run-time
- does not rely on a pre-defined multiresolution representation from which to build the adaptive triangle meshes each frame
- accounts for multiple view-dependent metrics
- ensures frame-to-frame coherence
- prevents discontinuities or cracks from appearing between meshes rendered at different resolutions.

Sophisticated computer programmers in this field have been able to use their own wave models on a host device that is synchronized to, and communicates with, an image generator. But now they can employ the same wave model on the image generator. The host device computes the hydrodynamic motion and frequency response of the ships, and communicates the wave model parameters to the image generator. The image generator, in turn, models and renders the exact same ocean waves. Consequently, the computed sway, surge, heave, yaw, pitch, and roll of the vessels are completely accurate within the context of the rendered ocean.

Future work needs to be done to address the issue of paging efficiency, regarding which ocean cells are paged in or out. Also, instead of using just the distance from the observer, the rendering engine could also use the field of view, view direction, and possibly the height above the water in its level of detail calculations.

Our solution was implemented using C++, Vega Prime, VSG, OpenGL, and Cg on a dual Intel Xeon processor (2.8 GHz) workstation and *n*VIDIA GeForce 7800 GTX graphics hardware. Our frame rate was consistently 60 Hz, and even when rendering the most complex ocean scenes never fell below 30 Hz.

AUTHOR BIOGRAPHY

Lawrence Lachman is a senior software engineer at MultiGen-Paradigm. He joined MultiGen-Paradigm in 1995 and has been the architect and project lead for Vega Prime Marine, a realtime ocean simulation module for Vega Prime, since 2002. His research has been primarily focused on simulating ocean waves, computer graphics, shader technology, and CPU optimization, and has led to the delivery of multiple first-to-market technologies. Prior to 2002 he was a charter member of the team that designed and developed FlightIG, and was one of the principal architects of the image generator.

Lawrence began his career with IBM in 1987. In eight years at IBM he designed and developed innovative software solutions in a product-oriented, R&D environment, and was awarded eight patents in the areas of multimedia, software engineering, computer security, and graphical user interface enhancement.

Lawrence received a B.A. in Computer Sciences from The University of Texas at Austin in 1991, where he completed graduate work in computer graphics that included advanced work in ray tracing.

REFERENCES

[1] M. A. Duchaineau, M.Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein, "ROAMing Terrain: Real-time Optimally Adapting Meshes." IEEE Visualization, Nov 1997, pp. 81–88.

[2] Mark Finch (2004), "Effective Water Simulation from Physical Models", GPU Gems, pp. 5–29.

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

[4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1994), Design Patterns, Addison-Wesley.

[5] Goda Y. (1990), "Random Waves and Spectra", "Handbook of Coastal and Ocean Engineering", Volume 1, Chapter 4, Edited by Herbich, J., Gulf Publishing Company.

[6] Markus Gross, R. Gatti, and Oliver G. Staadt, "Fast Multiresolution Surface Meshing", IEEE Visualization, October. 1995, pp. 135–142.

[7] Damien Hinsinger, Fabrice Neyret, Marie-Paule Cani, "Interactive Animation of Ocean Waves", ACM-SIGGRAPH/EG Symposium on Computer Animation, July 2002

[8] Yuri Kryachko (2005), **"**Using Vertex Texture Displacement for Realistic Water Rendering". GPU Gems 2, pp. 283–294.

[9] Lawrence M. Lachman, "Surf Zone Modeling for an EFV Trainer for the USMC", *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2006.* Paper No. 2816.

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

[11] 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*, August 1981, pp.317-324.

[12] Kathryn McCreight (1998), "A Note on the Selection of Wave Spectra for Design Evaluation", Naval Surface Warfare Center, Bethesda, Maryland, CRDKNSWC-HD-974-02.

[13] Jason L. Mitchell, "Real-Time Synthesis and Rendering of Ocean Water", ATI Research Technical Report, April 2005

[14] Ken Perlin, "An Image Synthesizer," SIGGRAPH 1985, pp. 287-296.

[15] Darwyn Peachey, "Modeling Waves and Surf," SIGGRAPH 1986, pp. 65-74.

[16] Jerry Tessendorf, "Simulating Ocean Water", *Simulating Nature: Realistic and Interactive Techniques Course Notes*, SIGGRAPH 2001.

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

[18] 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.

[19] William Van Dorn (1994), Oceanography and Seamanship. Second edition, Cornell Maritime Press.

[20] Tim Woodard, "GPU-Based Wave Simulation," *Proceedings of the IMAGE 2006 Conference*.