OceanSim

Vîlcu Valentin-Mihai 343
Iordache Alexandru-Stefan 343
Goidan Matei-Constantin 343
Ilie Florin Alexandru 332
Ilie George Cristian 332

Abstract

A real-time, tileable ocean surface that feels both physically plausible and artistically controllable is a core requirement for contemporary open-world and marine-themed games. (1) In this documentation we describe a Unity-based implementation of an ocean simulation driven by a Fast Fourier Transform (FFT) height-field, coupled with buoyancy sampling that allows dynamic objects—most notably boats—to interact convincingly with the water. The system synthesises an initial Phillips spectrum, evolves it in the frequency domain, and performs a two-dimensional inverse FFT each frame to obtain displacements, normals, and colour gradients. Our implementation leverages Unity Jobs and the Burst compiler to exploit CPU-side data parallelism on modern multi-core machines, achieving acceptable frame rates for moderate grid sizes. (2) Although architected for eventual migration to a GPU compute pipeline, the current prototype executes entirely on the CPU, which remains the principal limitation.

1 Introduction

Realistically rendering large expanses of ocean in real-time poses a unique challenge: the water surface must exhibit high-frequency capillary ripples, coherent swell patterns, and non-repeating tiling, all while allowing interactive objects to float and respond naturally. Established offline techniques—such as spectral synthesis of wave states—offer visual fidelity but historically exceeded the performance budgets of game engines. Leveraging the spectral approach popularised by Tessendorf (J.Tessendorf, 2001), and more recently adapted for GPU compute shaders (M.Finch, 2015).

2 Related Work

Early maritime renderers relied on Gerstner wave summations for their analytic simplicity, albeit at the expense of realism when combining many waves. Tessendorf's seminal work on FFT-based ocean surfaces demonstrated that the evolution of the frequency domain could efficiently recover short and long wavelength behavior. Subsequent real-time adaptations targeted GPU architectures through fragment programs and tile-based LOD strategies. More recent contributions integrate spray, foam, and ship-wake interactions, while others explore neural up-sampling of coarse simulations. Our approach inherits the spectral foundation but is distinguished by its exclusive use of Unity Jobs, permitting deployment on devices lacking robust compute capabilities.

3 Method

Our pipeline proceeds through four stages every frame: spectrum initialisation, frequency-domain evolution, inverse FFT, and mesh deformation. Each step is threaded using Unity's Job System; data are stored in NativeArray structures to maximise Burst compilation.

3.1 Spectrum Initialization

A discrete grid of $N \times N$ spectral samples represents complex wave amplitudes $H_0(k)$ at time $t = 0$. For each wavenumber $k = (k_x, k_z)$ we draw a Gaussian random pair (r_1, r_2) and scale by the square root of the Phillips spectrum $P(k)$ (Equation 1) to impose wind-aligned energy distribution:

$$P(k) = A * exp(\frac{\frac{-1}{|k|^2 * L^2}}{|k|^4 * ((k * w^2)^2}$$

where A is a tunable constant, $L = \frac{W^2}{g}$ is the largest supported wavelength determined by wind speed W, and l is a damping term suppressing capillary waves. We pre-compute the dispersion frequency $\omega(k) = \sqrt{g * |k|}$ and store arrays of twiddle factors for later FFT passes.

3.2 Temporal Evolution

At run-time, a **SpectrumJob** (Burst-compiled IJobParallelFor) updates each spectral coefficient according to the deep-water dispersion relation:

$$H(k,t) = H_0(k) * e^{i*\omega*t} + e^{-i*\omega*t}$$

To support horizontally *"chopped"* waves that displace vertices in x and z, we also compute slope fields $H_x(k,t)$ and $H_z(k,t)$ using the imaginary component scaled by $\frac{1}{|k|}$.

3.3 Inverse FFT

The three complex grids—height H_t, slope-x H_x, slope-z Hz — are each subjected to a 2-D inverse FFT. We implement a radix-2 Cooley–Tukey algorithm with bit-reversed reordering, cached twiddle tables, and in-place butterfly operations. The transform is executed row-wise followed by column-wise, operating entirely on CPU cache-friendly arrays.

3.4 Mesh Displacement and Shading

The spatial domain results are applied to a $(N+1) \times (N+1)$ vertex grid forming a single Unity mesh. For vertex (x, z) we sample periodic indices into the $H_t/H_x/H_z$ arrays, yielding:

$$\begin{bmatrix} v.x \\ v.y \\ v.z \end{bmatrix} = \begin{bmatrix} x + H_x \cdot \text{heightScale} \cdot \text{chopScale} \\ H_t \cdot \text{heightScale} \\ z + H_z \cdot \text{heightScale} \cdot \text{chopScale} \end{bmatrix}$$

Colours are interpolated between a deep-water tint and foam-white based on normalised height, producing intuitive white-cap highlights under high winds. Normals are recomputed in local space via forward differences and transformed into world space for lighting.

Patch size 50, res 128

Patch size 100, res 128

Patch size 100, res 256

3.5 Boat Floating Mechanics

Gameplay objects query the water surface through two public sampling methods:

$SampleHeight(Vector3, worldPos)$ – bilinear-interpolates H_t to return the water height at an arbitrary world-space XZ.

$SampleNormal(Vector3worldPos)$ – interpolates H_x/H_z to compute the local normal, facilitating tilt and damping computations.

Patch size 100, res 256, wind speed 30

A rigid-body boat applies buoyant forces at multiple probes distributed along its hull.

4 Future Work

GPU Compute Migration. The dominant bottleneck—two 2-D inverse FFT passes per frame—lends itself to compute shaders. A prototype indicates a potential 10× speed-up on mainstream GPUs.

Level-of-Detail (LOD). Nested quadtree tiles with varying grid resolutions could maintain high fidelity near the camera while amortising far-field cost.

Spectral Foam and Spray. Advection of vorticity fields and threshold-based foam masks would enrich visual complexity.

5 Conclusion

We have detailed a CPU-parallel FFT ocean surface integrated with buoyancy sampling for interactive boats. The system achieves 60 fps on modern desktop CPUs at 128^2 resolution while providing physically grounded wave spectra and artist-friendly controls. Although current performance scales poorly beyond 256^2 due to CPU cache constraints, the architecture forms a solid basis for subsequent GPU migration and gameplay extensions.

Limitations

CPU-Bound Implementation. All spectral evolution and FFT passes run on the CPU rather than the GPU, resulting in limited scalability for high-resolution grids or densely tiled scenes.

Deep-Water Assumption. The Phillips spectrum ignores shallow-water dispersion, precluding accurate coastal shoaling.

No Wave–Object Feedback. Boats influence neither the spectrum nor local wave heights, which may reduce realism in heavy seas.

References

1. I tried simulating the entire ocean.

2. How games fake water.

J.Tessendorf. 2001. Simulating ocean water. *Siggraph Course Notes*.

M.Finch. 2015. Real-time water rendering on gpu. *Game Developers Conference*.