

Article

Real-Time Wave Simulation of Large-Scale Open Sea Based on Self-Adaptive Filtering and Screen Space Level of Detail

Xi Duan, Jian Liu and Xinjie Wang *

Department of Computer Science and Technology, Ocean University of China, Qingdao 266000, China

* Correspondence: wangxinjie@ouc.edu.cn

Abstract: The real-time simulation technology of large-scale open sea surfaces has been of great importance in fields such as computer graphics, ocean engineering, and national security. However, existing technologies typically have performance requirements or platform limitations, and the two are often impossible to balance. Based on the camera-view-based screen space level of detail strategy and virtual camera pose adaptive filtering strategy proposed in this article, we have developed a fast and comprehensive solution for rendering large-scale open sea surfaces. This solution is designed to work without the need for special hardware extensions, making it easy to deploy across various platforms. Additionally, it enhances the degrees of freedom of virtual camera movement. After conducting performance tests under various camera poses, our filtering strategy was found to be effective. Notably, the time cost of simulation using 60 waves at the height of 6 m above sea level was only 0.184 ms. In addition, we conducted comparative experiments with four state-of-the-art algorithms currently in use, and our solution outperformed the others with the best performance and suboptimal visual effects. These results demonstrate the superiority of our approach in terms of both efficiency and effectiveness.

Keywords: computer graphics; ocean simulation; real-time rendering



Citation: Duan, X.; Liu, J.; Wang, X. Real-Time Wave Simulation of Large-Scale Open Sea Based on Self-Adaptive Filtering and Screen Space Level of Detail. *J. Mar. Sci. Eng.* **2024**, *12*, 572. <https://doi.org/10.3390/jmse12040572>

Academic Editor: Fabio Bruno

Received: 5 March 2024

Revised: 22 March 2024

Accepted: 25 March 2024

Published: 28 March 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the widespread application of ocean engineering (such as in navigation systems [1] or marine simulators [2]), scientific visualization, film visual effects, video games, and other fields, the importance of real-time rendering of large-scale open sea scenes has significantly increased in recent years. The challenge at hand is to simulate and render large-scale ocean surfaces while maintaining a balance between efficiency and quality on power-limited platforms like web and mobile devices. Existing industrial solutions for addressing ocean surface rendering issues primarily involve techniques such as Fast Fourier Transform (FFT) for wave simulation [3], which relies on bandwidth and general-purpose computing, Gerstner wave stacking technology [4], which is constrained by iteration times, and the method of bump mapping [5], which often lacks rendering details.

Eric Bruneton et al. proposed a GPU (Graphic Processing Unit) implementation of ocean rendering in their work [6], which involves the analytical calculation of the normal of the ocean surface height field. They also introduced smooth filtering based on a hierarchical representation of mixing geometry. However, their method lacks dynamic filtering strategies for camera roaming, and the screen space strategy they employed faces challenges in handling changes in camera pose. In response to these limitations, we have implemented several improvements to address these issues. In our approach, considering the requirements for cross-platform compatibility, ease of deployment, and the need to balance performance and effectiveness in large-scale ocean surface rendering, we have designed a forward rendering solution without using the General-Purpose Graphics Processing Unit (GPGPU). This approach relies solely on the basic programmable graphics pipeline to avoid multiple passes and does not depend on any other specialized hardware

extensions or compute shaders for non-graphics-related tasks. Additionally, we have utilized screen space level of detail and a self-adaptive filtering strategy to achieve a balance between performance and rendering quality. Our method offers advantages such as extremely low time cost and relatively high quality, making it suitable for applications in the field of ocean engineering.

Our main contributions are:

- In Section 3.2, we present a screen space level of detail strategy based on camera perspective and a novel adaptive filtering strategy based on virtual camera pose;
- We propose a large-scale open sea surface modeling solution that does not rely on special hardware extensions. Our pipeline is lightweight and handy for deploying. It is described in detail in Section 3.3;
- We have increased the degree of freedom of virtual camera motion under the screen space strategy, as discussed in Section 3.2.

A comparative analysis of our proposed method against four existing solutions in terms of efficiency has been conducted, specifically focusing on the time required for ocean surface height field modeling per frame. This comparison serves to highlight the advancements achieved through the implementation of our approach. Additionally, comprehensive tests have been performed to assess the modeling time cost at various camera heights, aiming to substantiate the efficacy of our level of detail and filtering strategy. The resulting renderings from different camera perspectives further underscore the enhanced degrees of freedom in camera movement facilitated by our methodology. To validate the usability of our system, we administered system usability scale questionnaires, the outcomes of which provide empirical support for the user-friendliness of our system. Detailed descriptions of our experimental procedures and findings can be found in Section 4.

2. Related Work

In this section, we will discuss four main topics: waveform simulation techniques for water rendering, level of detail schemes, modeling schemes, and ocean engineering applications. These areas cover important techniques and practical applications in the field of water rendering and simulation, providing a comprehensive perspective to understand and explore the latest developments in the field of water simulation.

2.1. Waveform Simulation Techniques for Water Rendering

The technique of simulating waveforms plays a crucial role in rendering dynamic water surfaces in the field of computer graphics. Various methods have been employed to achieve this, including linear wave superposition, particle-based water simulation, physics-based approaches, pre-rendering, and statistical models. Linear wave superposition involves the accumulation of different linear wave functions to construct the undulating surface, which is particularly effective for deep water. One commonly used method is the Gerstner wave introduced by Gerstner in 1802 [7]. Particle-based water simulation, initially proposed by Yuksel in 2007 [8] and improved by Jeschke et al. [9,10], uses particles to represent individual water waves, enabling wave reflection and interaction with dynamic objects. This approach simplifies the dynamic simulation of three-dimensional water waves to the level of simulating a particle system moving on a plane. Physics-based methods [11] simulate water by solving the Navier–Stokes equation, suitable for offline rendering. Pre-rendering methods [12,13] involve the prior calculation and storage of water surface wave information, commonly employed in applications requiring high realism. Statistical model methods [14,15], based on FFT, find widespread use in the film and video game industries [16] due to their exceptional realism and global controllability.

2.2. Level of Detail Schemes

LOD (Level of detail), an essential technique in real-time computer graphics, adjusts detail levels at different distances and viewpoints to enhance performance and reduce computational load. Common LOD implementation methods include Geometry Clipmap [17],

CDLOD (Continuous Distance-Dependent Level of Detail) [18], and Quadtree [6,19]. Geometry Clipmap utilizes nested regular grids to maintain a seamless and continuous terrain surface, allowing dynamic clipping and the addition of geometric details for improved real-time performance. CDLOD is a real-time continuous detail-level rendering method, avoiding discontinuities between different LODs. Quadtree, a common LOD implementation, dynamically adjusts detail levels at different distances through a hierarchical data structure, providing flexibility and simplicity in LOD schemes.

2.3. Modeling Schemes

Modeling schemes encompass manual modeling, screen space mesh projection, and multi-resolution patch systems. Manual modeling involves adjusting water shape, ripples, and textures using professional tools, and is suitable for static scenes and cases requiring artistic control. Screen space mesh projection [20] employs screen space techniques for real-time water simulation, which is particularly suitable for real-time applications but has limited flexibility. Multi-resolution patch systems [6,21] represent the water surface using patches of varying resolutions, optimizing performance by using low-resolution patches in the distance and high-resolution patches up close to ensure finer details and realism.

2.4. Ocean Engineering Applications

As information technology advances rapidly, water rendering technology finds significant applications in engineering across various domains. In educational platforms, the technology enhances the learning experience for engineering students, providing realistic scenarios for water resource management and hydraulics experiments [22]. In Geographic Information Systems (GIS) [23], water rendering supports map-making and spatial analysis by providing clear visualizations of geographical information such as water bodies and flows. The applications in the field of marine and ocean engineering are focused on the physical modeling of the waves, while rendering technology is essential for visualizing simulation results. In ship engineering [24], water rendering aids in visualizing ship motion and interaction effects in different water bodies, facilitating a deeper understanding of ship performance for design optimization. In fluid dynamics [25], water rendering technology is applied to visualize fluid flow data, aiding engineers in analyzing water flow patterns and optimizing hydraulic structures. In deep-sea petroleum extraction engineering [26], water rendering technology is crucial for simulating water environments, providing real-time ocean data visualization for platform design, operation, and safety analysis. These applications underscore the diverse value of water rendering technology in engineering, offering powerful tools and methods to drive development and innovation in the field.

3. Methodology

The schematic diagram in Figure 1 illustrates the fundamental workflow of our methodology. The diagram is divided into two main sections. The left side delineates the tasks executed on the CPU host, encompassing the generation of wave data utilizing a spectrum formula [27], computation of transformation matrices for subsequent GPU processing, incorporation of camera specifics, and initialization of essential global variables crucial for modeling and shading, such as lighting parameters and temporal settings. On the right side, we elucidate the modeling pipeline executed on the GPU device, which encompasses four primary components. Firstly, in Section 3.1, we present the Gerstner wave model derived from the simplified Euler equation, serving as the theoretical foundation for our wave simulation approach. Secondly, we delve into the self-adaptive screen space mesh generation detailed in Section 3.2, functioning within the vertex shader stage and tessellation control shader stage. Furthermore, Section 3.3 represents the crux of our methodology, introducing an innovative screen space adaptive filtering strategy to render animated waves at varying levels of detail. Lastly, we offer a concise overview in Section 3.4 outlining our shading strategy, which, while not the primary focus of our research, plays a pivotal role in ensuring the seamless operation of the entire system.

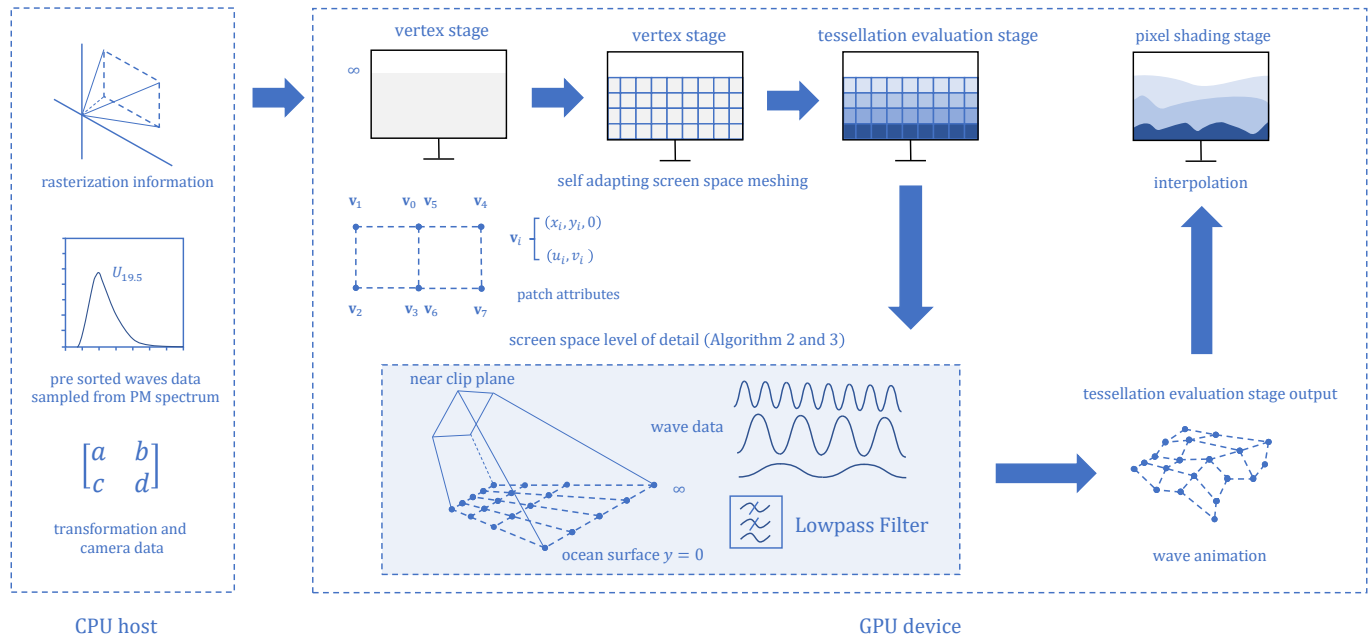


Figure 1. The basic flow path of our method.

3.1. Wave Model

The wave model begins with the Euler Equations of an incompressible fluid:

$$\begin{aligned} \frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla) \mathbf{v} &= -\nabla p + \mathbf{F} \\ \nabla \cdot \mathbf{v} &= 0 \end{aligned} \quad (1)$$

where $\mathbf{v}(\mathbf{x}, t)$ represents the velocity field, $\mathbf{x}$ represents position, t is time, p represents hydrostatic pressure, and $\mathbf{F}$ represents external force. $\mathbf{F}$ can be represented as a gradient of some scalar field if it is conservative. In addition, for waves that have movements caused by a conservative force, we have $\oint_L \mathbf{v} \cdot d\mathbf{l} = 0$, and the velocity field $\mathbf{v}$ can be also represented as the gradient of some scalar function $\varphi(\mathbf{x}, t)$ where $\mathbf{x} = (x, y, z)$ represent field position. Furthermore, after linearization, if we assume $\mathbf{F}$ is just gravity, and consider only the changing shape of the water surface, this simplification allows us to neglect the complex ocean appearance such as surf breaking with bubbles; underwater motion can also be ignored. This focused approach enables us to derive the following simplified model:

$$\begin{aligned} \frac{\partial \varphi}{\partial t} &= -p + p_a - gy \\ \nabla^2 \varphi &= 0 \end{aligned} \quad (2)$$

Since we constrain wave movements on the ocean surface, we have $p = p_a$, where p_a is atmospheric pressure. After differentiating the first equation by t , we obtain $\frac{\partial^2 \varphi}{\partial t^2} = -g \frac{\partial h}{\partial t}$. and by setting the kinematic term of surface $\frac{\partial y}{\partial t} = v_y$, we obtain:

$$\begin{aligned} \frac{\partial^2 \varphi}{\partial t^2} + g \frac{\partial \varphi}{\partial y} &= 0 \\ \nabla^2 \varphi &= 0 \end{aligned} \quad (3)$$

By using bottom boundary conditions $\frac{\partial \varphi}{\partial y} = 0$ in $y = -\eta$ and a free-surface boundary condition $\frac{\partial^2 \varphi}{\partial t^2} + g \frac{\partial \varphi}{\partial y} = 0$ in $y = 0$ to solve $\nabla^2 \varphi = 0$ [28], we obtain:

$$\varphi = \frac{gA}{\omega} \sin(\mathbf{k} \cdot \mathbf{x} - \omega t) \frac{\cosh(\|\mathbf{k}\|(y + \eta))}{\cosh(\|\mathbf{k}\|\eta)} \quad (4)$$

In Equation (4), A denotes the amplitude, ω represents the angular frequency, and $\mathbf{k}$ signifies the wave vector. As depth $\eta \rightarrow \infty$, the velocity potential would be:

$$\varphi = \frac{gA}{\omega} \sin(\mathbf{k} \cdot \mathbf{x} - \omega t) e^{\|\mathbf{k}\|y} \quad (5)$$

At the ocean surface, where $y = 0$, the surface height field can be determined as follows:

$$\begin{aligned} \varphi &= \frac{gA}{\omega} \sin(\mathbf{k} \cdot \mathbf{x} - \omega t) \\ h &= -\frac{1}{g} \frac{\partial \varphi}{\partial t} \end{aligned} \quad (6)$$

To achieve the appearance of choppy waves in practical applications, it is advisable to introduce a slight offset along the gradient direction of the height field $h(\mathbf{x}_\perp, t)$, where $\mathbf{x}_\perp = (x, z)$, as suggested by [3]. Consequently, we have:

$$\Delta \mathbf{x}_\perp = \frac{1}{\|\mathbf{k}\|} \nabla h(\mathbf{x}_\perp, t) = \frac{1}{\|\mathbf{k}\|} \begin{bmatrix} -k_x A \sin(\mathbf{k} \cdot \mathbf{x} - \omega t) \\ -k_z A \sin(\mathbf{k} \cdot \mathbf{x} - \omega t) \end{bmatrix} \quad (7)$$

where $\Delta \mathbf{x}_\perp$ represents the offset of the horizontal projection of the ocean surface position $\mathbf{p}$ concerning the horizontal projection of field position $\mathbf{x}_\perp$. Combining $\Delta \mathbf{x}_\perp$ and $h(\mathbf{x}_\perp, t)$, we obtain $\Delta \mathbf{x} = (\Delta x_x, h(\mathbf{x}_\perp, t), \Delta x_z)^T$. Therefore, the parametric equation for the ocean surface position $\mathbf{p}$ can be expressed as follows:

$$\mathbf{p} = \mathbf{x}|_{y=0} + \Delta \mathbf{x} = \begin{bmatrix} x \\ 0 \\ z \end{bmatrix} + \begin{bmatrix} -\frac{k_x}{\|\mathbf{k}\|} A \sin(\mathbf{k} \cdot \mathbf{x} - \omega t) \\ A \cos(\mathbf{k} \cdot \mathbf{x} - \omega t) \\ -\frac{k_z}{\|\mathbf{k}\|} A \sin(\mathbf{k} \cdot \mathbf{x} - \omega t) \end{bmatrix} \quad (8)$$

Equation (8) demonstrates that the shape of waves can be effectively modeled using trochoids. Utilizing the Gerstner wave model $\mathbf{p}(\mathbf{x}, t)$, it becomes straightforward to determine the height offset of each surface position relative to the horizontal plane $y = 0$ in world space. Additionally, this model facilitates the accurate calculation of the normal vector representation.

$$\mathbf{n} = \text{normalize} \left(\frac{\partial \mathbf{p}}{\partial x} \wedge \frac{\partial \mathbf{p}}{\partial z} \right) \quad (9)$$

3.2. Self-Adapting Screen Space Meshing

In this section, our objective is to transform the wave model into a mesh representation. The desired outcome is to achieve the appearance of an infinite ocean surface from any viewpoint, provided that $y > 0$ and there is no roll component in the camera's orientation (the treatment of the camera's roll component is specifically addressed in Section 3.2). Before proceeding, let us define some fundamental symbols: $\mathbf{P}$, $\mathbf{V}$ denote the projection matrix and the camera view matrix, respectively; homogeneous coordinates $\mathbf{v}$, with $z = 0$, are used to represent a vertex point in screen space; and $\mathbf{p}$ is used to denote a mesh vertex in world space. Taking perspective division into account, the point at $z = \infty$ along the z -axis in world space can be depicted as follows:

$$\mathbf{x}_\infty = (0 \quad 0 \quad -1 \quad 0)^T \quad (10)$$

Assuming $f = \infty$, namely the far clip plane distance of $\mathbf{P}$ is infinity, the projected position in NDC (Normalized Device Coordinate) space can be expressed as follows:

$$\mathbf{v}_{\infty} = \mathbf{P}\mathbf{V}\mathbf{x}_{\infty} \quad (11)$$

After perspective division, the y component of $\mathbf{v}_{\infty}$, denoted as $H \in [-1, 1]$, represents the maximum height of the projected grid in screen space, as illustrated in Figure 2b. To accommodate adaptive resolution, we leverage the hardware tessellation feature to tessellate the area of the screen that is occupied. Initially, we start with two patches comprising eight vertices $\{\mathbf{v}_i\}$, which cover the screen's NDC (Normalized Device Coordinates) area of $[-1, 1]^2$. Given that the camera motion lacks a roll component, the projected grid area must form a rectangle. Consequently, we can straightforwardly adjust the y component of the four upper vertices of the two rectangles to H .

$$\mathbf{v}_i = (v_{i,x} \ H \ 0 \ 1)^T, \quad i \in \{0, 1, 4, 5\} \quad (12)$$

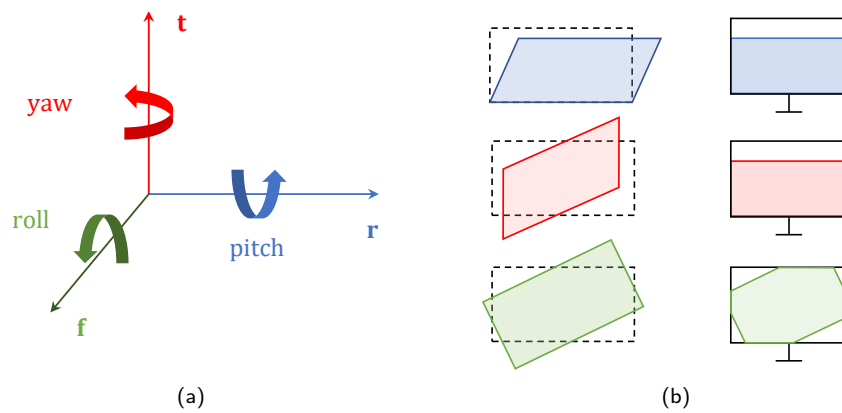


Figure 2. Euler angles and projection plane. (a) Camera basis and Euler angle. (b) Projection plane under different camera movement.

We have prepared two rectangular patches for tessellation. To implement tessellation, we utilize OpenGL's control shader and evaluation shader. A self-adaptive tessellation strategy is crucial to ensure that we do not generate an overly fine-grained mesh when the projected area is small. This situation often arises when the angle between the view direction and the $y = 0$ plane approaches zero. Let us define N (with $\sup N$ dependent on the hardware device, typically $\sup N \leq 64$) as the subdivision count, which represents the maximum number of cells that can be subdivided along the width of the rectangle. The self-adaptive subdivision count can then be calculated as follows:

$$\left\lfloor \frac{(H+1)N}{2} \right\rfloor \quad (13)$$

The OpenGL control shader plays a pivotal role in adjusting the tessellation level of detail within a quad domain by modifying its inner and outer levels. Following this, the evaluation stage of the pipeline subdivides the two patches into the desired triangle mesh. After tessellation, we obtain the positions of vertices $\{\mathbf{x}'_i\}$, each with a z component of 0. The subsequent step involves implementing inverse projection to transform these vertices from NDC (Normalized Device Coordinates) space back to world space. Initially, it is important to note that the projected vertices $\{\mathbf{b}'_i\}$ located on the far clip plane have a z value of 1.

$$\mathbf{b}'_i = (x'_x \ x'_y \ 1 \ 1)^T, \quad i \in \{0, 1, 4, 5\} \quad (14)$$

The direction vector from the camera position $\mathbf{o}$ to $\mathbf{x}'_i$ should be:

$$\mathbf{d}_i = \frac{\mathbf{V}^{-1}\mathbf{P}^{-1}\mathbf{b}'_i - \mathbf{o}}{\|\mathbf{V}^{-1}\mathbf{P}^{-1}\mathbf{b}'_i - \mathbf{o}\|} \quad (15)$$

The distance from $\mathbf{o}$ to the horizontal plane at $y = 0$ along $\mathbf{d}_i$ is represented as t_i .

$$t_i = \frac{o_y}{|\mathbf{d}_{i,y}|} \quad (16)$$

So the intersection, namely inverse projected vertex position $\mathbf{x}_i$ can be represented as:

$$\mathbf{x}_i = \mathbf{o} + t_i \mathbf{d}_i \quad (17)$$

Following the application of inverse projection to all tessellated screen space triangles, they can be manipulated just like any conventional mesh.

Roll Component Processing

Euler angles are commonly used to represent rotations for rigid bodies in engineering practice due to their direct nature. Given a local coordinate basis $\{\mathbf{r}, \mathbf{t}, \mathbf{f}\}$, it is straightforward to decompose a rotation into three components: pitch, roll, and yaw. These components describe the counterclockwise rotation, in radians, along each axis (ψ, θ, φ) . However, it is worth noting that Euler angles do not align with the convention adopted in ocean engineering and the marine sector in general [29]. To address this, we can use Equation (18) to facilitate the mutual conversion between Euler angles (ψ, θ, φ) and quaternions (q_w, q_x, q_y, q_z) . Additionally, in engineering practice, when θ approaches $\frac{\pi}{2}$, we can adjust θ by $\pm\delta\theta$ to avoid the issue of gimbal lock.

$$\begin{aligned} \begin{bmatrix} \varphi \\ \theta \\ \psi \end{bmatrix} &= \begin{bmatrix} \text{atan2}(2(q_w q_x + q_y q_z), 1 - 2(q_x^2 + q_y^2)) \\ \arcsin(2(q_w q_y - q_z q_1)) \\ \text{atan2}(2(q_w q_z + q_x q_y), 1 - 2(q_y^2 + q_z^2)) \end{bmatrix} \\ \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} &= \begin{bmatrix} \cos(\varphi/2) \cos(\theta/2) \cos(\psi/2) + \sin(\varphi/2) \sin(\theta/2) \sin(\psi/2) \\ \sin(\varphi/2) \cos(\theta/2) \cos(\psi/2) - \cos(\varphi/2) \sin(\theta/2) \sin(\psi/2) \\ \cos(\varphi/2) \sin(\theta/2) \cos(\psi/2) + \sin(\varphi/2) \cos(\theta/2) \sin(\psi/2) \\ \cos(\varphi/2) \cos(\theta/2) \sin(\psi/2) - \sin(\varphi/2) \sin(\theta/2) \cos(\psi/2) \end{bmatrix} \end{aligned} \quad (18)$$

As depicted in Figure 2b, it is evident that the camera movement involving yaw and pitch components does not impact the horizontal orientation of the projection plane. Utilizing Euler angles, any arbitrary rotation matrix $\mathbf{R}$ in $\mathbb{R}^3$ can be expressed as:

$$\mathbf{R} = \mathbf{R}_f(\varphi) \mathbf{R}_t(\theta) \mathbf{R}_r(\psi) \quad (19)$$

Since matrix multiplication adheres to the associative law, we have the flexibility to postpone the roll component $\mathbf{R}_f(\varphi)$ until after the projected vertices' offset, denoted as $\mathbf{p}_i$, has been performed. This implies that we should separate $\mathbf{R}_f$ from the composite rotation matrix $\mathbf{R}$.

$$\varphi = \text{atan2}\left(\frac{R_{21}}{\cos\theta}, \frac{R_{11}}{\cos\theta}\right), \quad R_{31} \neq \pm 1 \quad (20)$$

Fortunately, in practice, our raw rotation data for camera movement is often represented by Euler angles or a quaternion rather than a rotation matrix. This means that φ is readily available to us, and Equation (20) is primarily for situations where Euler angles are not accessible to developers, such as in shader redevelopment. The final issue pertains to the local coordinate basis, which involves $\mathbf{R}_f(\varphi)$ describing the rotation by the base vector $\mathbf{f}$, namely the forward vector of the camera. Consequently, the roll matrix should be computed using Rodrigues' rotation formula:

$$\mathbf{R}_f(\varphi) = \cos\varphi \mathbf{I} + (1 - \cos\varphi) \mathbf{f} \mathbf{f}^T + \sin\varphi \mathbf{f}^\wedge \quad (21)$$

where $\mathbf{f}^\wedge$ represents the skew-symmetric matrix constructed by $\mathbf{f}$. $\mathbf{R}_f(\varphi)$ should be calculated in the CPU host and then be transmitted into the GPU device as global data for proficiency consideration. We use Equations (19) and (20) to decompose roll component out of camera matrix V' used in Algorithm 1 and the shading part, and we use Equation (21) to construct V used in Section 3.2, Algorithms 1 and 2.

Algorithm 1: Calculate Filter Threshold.

Input: View direction $\mathbf{d}$, view matrix $\mathbf{V}$ without roll component, camera position $\mathbf{o}$, distance n from camera to near clip plane, half fov θ , subdivision count N

Output: Vertical length L for current grid cell

```

1 calculate filter threshold;
2  $\mu \leftarrow n \tan \theta / N$ ;
3  $\mathbf{t} \leftarrow (0, 1, 0, 0)^T$ ;
4  $\mathbf{f} \leftarrow (0, 0, -1, 0)^T$ ;
5  $\mathbf{du} \leftarrow \mu \mathbf{t}$ ;
6  $\mathbf{d}^{\text{view}} \leftarrow \mathbf{V} \mathbf{d}$ ; // transform  $\mathbf{d}$  into camera space
7  $\mathbf{d}_\perp \leftarrow (0, d_y^{\text{view}}, d_z^{\text{view}}, 0)$ ; //  $\mathbf{d}_\perp$  and  $\mathbf{d}_i$  in Figure 3 are equivalent
8  $\mathbf{q}_\perp \leftarrow n \mathbf{d}_\perp / (\mathbf{d}_\perp \cdot \mathbf{f}) + \mathbf{du}$ ; //  $\mathbf{q}_\perp$  and  $\mathbf{q}_i$  in Figure 3 are equivalent
9  $\mathbf{d}' \leftarrow \mathbf{V}^{-1} \mathbf{d}_\perp / \|\mathbf{V}^{-1} \mathbf{d}_\perp\|$ ; // transform  $\mathbf{d}_\perp$  back to world space
10  $\mathbf{q}' \leftarrow \mathbf{V}^{-1} \mathbf{q}_\perp / \|\mathbf{V}^{-1} \mathbf{q}_\perp\|$ ; // transform  $\mathbf{q}_\perp$  back to world space
11  $\alpha \leftarrow |o_y / d'_y|$ ;
12  $\beta \leftarrow |o_y / q'_y|$ ;
13  $L \leftarrow \|\beta \mathbf{q}' - \alpha \mathbf{d}'\|$ ;
14 return  $L$ 

```

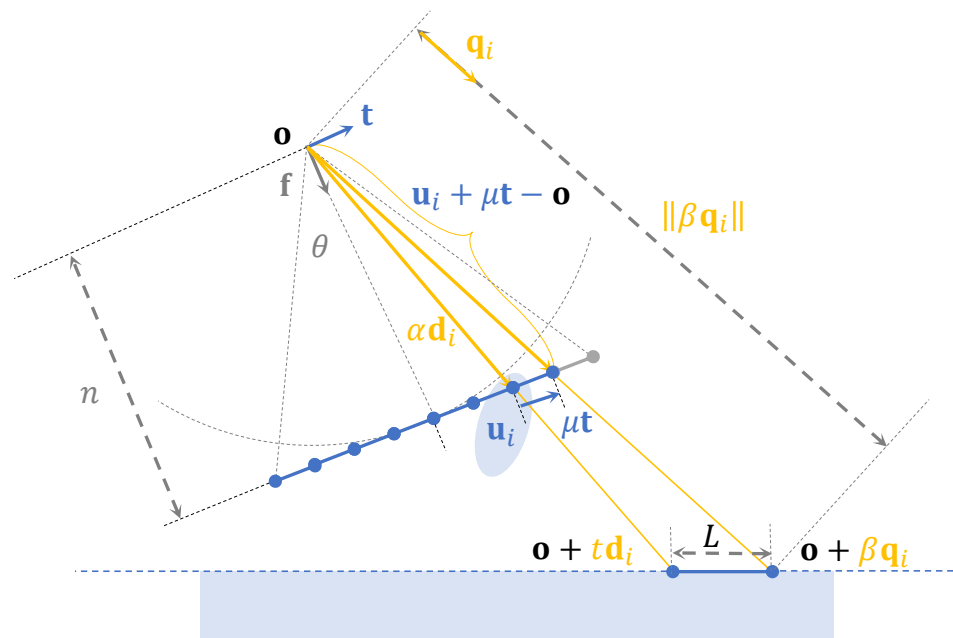


Figure 3. Algorithm for calculating vertical grid length of each screen space vertex.

Algorithm 2: Meshing and Inverse Projection.

Input: View matrix $\mathbf{V}$ without roll component and $\mathbf{V}'$ with roll component, camera position $\mathbf{o}$, distance n from camera to near clip plane, half fov θ , time t , projection matrix $\mathbf{P}$, screen space patch $\{\mathbf{v}_i\}$, subdivision count N , Gerstner waves $\mathcal{W}$, Nyquist sampling frequency k

Output: Set $\mathcal{V}$ of attributes to be interpolated between mesh vertices

```

1 calculate projected height  $H \in [-1, 1]$  for horizon; // in vertex shader
2  $\mathbf{x}_\infty \leftarrow (0, 0, -1, 0)^T$ ;
3  $\mathbf{v}_\infty \leftarrow \mathbf{P}\mathbf{V}\mathbf{x}_\infty$ ;
4  $H \leftarrow x_{\infty,y} / x_{\infty,w}$ ;
5 prepare quad to be subdivided;
6 foreach patch do
7   for  $i \leftarrow 0$  to 3 do
8     if  $i \in \{0, 1\}$  then
9        $v_{i,y} \leftarrow H$ 
10    else
11       $v_{i,y} \leftarrow 0$ 
12  $H \leftarrow \max(H + 1, 1)$ ;
13 apply tessellation granularity  $H \in [0, 1]$ ; // in control shader
14 foreach patch do
15   TessLevelInner0  $\leftarrow N$ ;
16   TessLevelInner1  $\leftarrow \lfloor NH \rfloor$ ;
17   for  $i \leftarrow 0$  to 3 do
18     if  $i \in \{0, 2\}$  then
19       TessLevelOuter $i$   $\leftarrow \lfloor NH \rfloor$ 
20     else
21       TessLevelOuter $i$   $\leftarrow N$ 
22  $N \leftarrow \lfloor NH \rfloor$ ;
23 interpolate vertices in screen space; // in evaluation shader
24  $\mathcal{V} \leftarrow \emptyset$ ;
25 foreach patch do
26   fetch  $(u, v)$ ,  $\{\mathbf{v}_i\}$  from patch;
27    $\mathbf{s} \leftarrow (1 - u)\mathbf{v}_0 + u\mathbf{v}_1$ ;
28    $\mathbf{t} \leftarrow (1 - u)\mathbf{v}_2 + u\mathbf{v}_3$ ;
29    $\mathbf{x}' \leftarrow (1 - v)\mathbf{s} + v\mathbf{t}$ ;
30    $\mathbf{b}' \leftarrow (x'_x, x'_y, 1, 1)^T$ ;
31    $\mathbf{d} \leftarrow \mathbf{V}^{-1}\mathbf{P}^{-1}\mathbf{b}' - \mathbf{o} / \|\mathbf{V}^{-1}\mathbf{P}^{-1}\mathbf{b}' - \mathbf{o}\|$ ;
32    $L \leftarrow \text{Calculate Filter Threshold}(\mathbf{d}, \mathbf{V}, \mathbf{o}, n, \theta, N)$ ; // call Algorithm 1
33    $t \leftarrow o_y / d_y$ ;
34    $\mathbf{x} \leftarrow \mathbf{o} + t\mathbf{d}$ ;
35    $(\Delta\mathbf{x}, \partial\mathbf{p}/\partial u, \partial\mathbf{p}/\partial v, \sigma) \leftarrow \text{Wave}(\mathcal{W}, \mathbf{x}, L, t, k)$ ; // call Algorithm 3
36    $\mathcal{V} \leftarrow \mathcal{V} \cup \{(\mathbf{x} + \Delta\mathbf{x}, \partial\mathbf{p}/\partial u, \partial\mathbf{p}/\partial v, \sigma)\}$ ;
37   use  $P\mathbf{V}'(\mathbf{x} + \Delta\mathbf{x})$  for position projection; //  $\mathbf{V}'$  has roll component
38 return  $\mathcal{V}$ 

```

Algorithm 3: Wave.

Input: Gerstner waves $\mathcal{W}$, vertex position of projected plane $\mathbf{p}$, vertical length L for current grid cell, time t , Nyquist sampling frequency k

Output: Wave offset $\Delta\mathbf{x}$, tangent vector $\partial\mathbf{p}/\partial u$, bitangent vector $\partial\mathbf{p}/\partial v$, height variance σ

```

// this function is called by Algorithm 2
1  $\Delta\mathbf{x} \leftarrow (0,0,0)^T$ ;
2  $\partial\mathbf{p}/\partial u \leftarrow (1,0,0)^T$ ;
3  $\partial\mathbf{p}/\partial v \leftarrow (0,0,1)^T$ ;
4  $\sigma \leftarrow (0,0)^T$ ;
5  $\mathbf{u} \leftarrow (\mathbf{p}_x, \mathbf{p}_y)^T$ ;
6  $g \leftarrow 9.81$ ;
7 foreach  $wave \in \mathcal{W}$  do
8   fetch  $\{\omega, \mathbf{k}, A\}$  from  $wave$ ;
9    $\lambda \leftarrow 2\pi g / \|\mathbf{k}\|$ ;
10  if  $|\lambda\omega| \geq kL$  then
11     $\varphi \leftarrow \mathbf{k} \cdot \mathbf{u} - \omega t$ ;
12    apply Equation (8) to obtain  $\Delta\mathbf{x}|_{wave}$ ;
13     $\Delta\mathbf{x} \leftarrow \Delta\mathbf{x} + \Delta\mathbf{x}|_{wave}$ ;
14    apply Equation (8) to obtain  $\partial\mathbf{p}/\partial u|_{wave}$ ;
15    apply Equation (8) to obtain  $\partial\mathbf{p}/\partial v|_{wave}$ ;
16     $\partial\mathbf{p}/\partial u \leftarrow \partial\mathbf{p}/\partial u - \partial\mathbf{p}/\partial u|_{wave}$ ;
17     $\partial\mathbf{p}/\partial v \leftarrow \partial\mathbf{p}/\partial v - \partial\mathbf{p}/\partial v|_{wave}$ ;
18    update  $\sigma$  by slope distribution of  $wave$ 
19 return  $\Delta\mathbf{x}, \partial\mathbf{p}/\partial u, \partial\mathbf{p}/\partial v, \sigma$ 

```

3.3. Screen Space Level of Detail

In the previous step, we obtained a flat triangle mesh plane, and the subsequent task is to displace each horizontal vertex by our wave model $\mathbf{p}(x, y, z)$. We must begin by generating wave data. A Gerstner wave is primarily composed of the following components: wavelength λ , which represents the distance over which the wave completes one full cycle; amplitude A , which denotes the maximum height of the wave; angular frequency ω , which refers to the phase addition per unit time; and wave vector $\mathbf{k}$, which describes the propagation direction of the wave. For optimization purposes, we generate these data [30] using random number generators on the CPU host only once at the beginning and deliver it, along with any other shading-related data, to the GPU device memory through an OpenGL uniform buffer. This eliminates the need for updating to be performed during every render loop.

The Pierson–Moskowitz spectrum, based on measurements of waves taken by accelerometers on British weather ships in the North Atlantic, is utilized [27]. The spectra are characterized by the following form:

$$S(\omega) = \frac{\alpha g^2}{\omega^5} \exp\left(-\beta\left(\frac{\omega_0}{\omega}\right)^4\right) \quad (22)$$

which shows the energy distribution of gravity waves as a function of their frequency, where $\alpha = 8.1 \times 10^{-3}$, $\beta = 0.74$, $\omega_0 = g/U_{19.5}$ and $U_{19.5}$ is the wind speed at a height of 19.5 m above the sea surface, the height of the anemometers on the weather ships used by Pierson and Moskowitz in 1964 [27].

Suppose we have N waves with pre-calculated parameters, each animated vertex $\mathbf{x}$ will be offset by $\sum_{i=1}^N \Delta \mathbf{x}_i$. Consequently, we obtain the animated vertex as follows:

$$\mathbf{p}_i = \mathbf{x}_i + \sum_{i=1}^N \Delta \mathbf{x}_i \quad (23)$$

However, implementing this concept directly on our grid may not be ideal. This is because our screen-space projected grid is regular and uniform, resulting in significant variation in the resolution of the inversely projected horizontal grid for different t_i . In particular, the difference in the stride length that a grid represents between the camera's near clip plane and far clip plane will be quite substantial. This discrepancy leads to a severe aliasing problem, as we attempt to sample waves across all frequencies with an irregular spatial sampling frequency, namely the inverse projected grid resolution. Low-pass filtering becomes essential. Given that we are using a free camera, the filtering strategy should be adaptive. We need to take into account the Nyquist sampling theorem:

$$f_s \geq 2f_{\max} \quad (24)$$

in Equation (24), f_s represents the sample frequency, and $f_{\max}$ denotes the maximum frequency of signal s . If the sampling frequency is too low, it fails to accurately reconstruct the true signal; conversely, if the sampling frequency is excessively high, it results in a large frequency resolution. In practical scenarios, the sampling frequency is typically chosen to be higher than twice the Nyquist frequency (often by a factor of k , where $k \in [3, 5]$). In our application, the sampling frequency, equivalent to the resolution of the inverse projected grid, remains fixed. To adhere to the Nyquist limit for each grid, we employ a low-pass filter for adaptation.

Back to physics, the mechanical wave parameter λ represents the propagation length of the wave in a full phase, ω represents the addition of phase per unit of time, and the wave speed v can be represented as:

$$v = \lambda \omega \quad (25)$$

v indicates how far a wave propagates per unit of time, and when we fix time, we have a low-pass filter:

$$v \leq \frac{L}{k} \quad (26)$$

The illustration of our adaptive algorithm is shown as Figure 3. Assuming we have a 2D camera atlas $\{\mathbf{o}, \{\mathbf{t}, \mathbf{f}\}\}$, L represent inverse projected grid length projected to $\mathbf{f}$, the viewing direction unit vector. The intersection point $\mathbf{u}_i$ between $\mathbf{d}_i$ (which is equivalent to $\mathbf{d}_{\perp}$ in Algorithm 1) and screen space projected grid can be calculated by solving equation $(\alpha \mathbf{d}_i - n \mathbf{f}) \cdot \mathbf{f} = 0$ for α .

$$\mathbf{u}_i = \mathbf{o} + \alpha \mathbf{d}_i \quad (27)$$

Since the screen space projected grid is uniform, using μ to represent the height of each screen space projected grid in world space, the offset of $\mathbf{u}_i$ along the top vector $\mathbf{t}$ can be represented as:

$$d\mathbf{u}_i = \mu \mathbf{t} = \frac{n \tan \theta}{N} \mathbf{t} \quad (28)$$

where $\theta = \frac{\text{fov}}{2}$, and we can easily obtain an auxiliary vector $\mathbf{q}_i$ (which is equivalent to $\mathbf{q}_\perp$ in Algorithm 1).

$$\mathbf{q}_i = \frac{\mathbf{u}_i + d\mathbf{u}_i - \mathbf{o}}{\|\mathbf{u}_i + d\mathbf{u}_i - \mathbf{o}\|} \quad (29)$$

In Algorithm 1, β shares the same principle as t , and the threshold length L can be calculated by:

$$L = \|\beta\mathbf{q}_i - t\mathbf{d}_i\| \quad (30)$$

The overall process of the modeling is shown in Algorithm 2, and the vertex attributes in the output $\mathcal{V}$ of Algorithm 2 will be interpolated by the GPU for shading in the pixel shader.

3.4. Rendering

Since our focus is on modeling, we employ only fundamental shading techniques for rendering. This includes adopting a constant value for sun radiance and utilizing environmental mapping instead of atmospheric scattering models for global illumination.

Although the Algorithm 2 outputs interpolated vertex attributes to the pixel shader, including $\mathbf{p}, \partial\mathbf{p}/\partial u, \partial\mathbf{p}/\partial v, \sigma$, we can still follow the approach of [6] to execute Algorithm 3 per pixel in the pixel shader, and we also follow the approach of [6] to combine three different ways of dealing with the effects of the waves on scene appearance: displacement maps, normal perturbation, and BRDF (Bi-directional Reflection Distribution Function) modification with the help of the GLSL (OpenGL Shading Language) function smoothstep and the threshold L . Although not as accurate as analytical calculations, in fact using attribute interpolation for coloring is still feasible. If interpolation is used for shading, more patches and vertices are needed to improve the grid resolution of screen space. At the same time, due to the issue of uniform grid resolution of screen space, certain hacks (such as volumetric fog) are needed to alleviate artifacts near the far clip plane.

Given the conversion of the filtering scale from grid to pixel, effectively resulting in an increased sampling rate, it is necessary for the filtering scale, denoted as P , to be reduced:

$$P = \frac{L}{\mu h} \quad (31)$$

The variable h signifies the height resolution of the screen and μh is the sampling frequency. Concurrently, given that the control flow on a GPU significantly depletes register resources, as highlighted in [31], an optimization strategy is adopted from [6,31]. This approach involves pre-sorting the waves on the CPU host and employing a threshold $T = Pk$ as the criterion for filtering levels. Subsequently, a mapping is utilized to directly ascertain the starting index, where k represents the Nyquist sampling frequency.

$$\left\lceil \log_2 \left(\frac{T}{\inf \{\lambda_i \omega_i\}} \right) \frac{C}{\log_2 \left(\frac{\sup \{\lambda_i \omega_i\}}{\inf \{\lambda_i \omega_i\}} \right)} \right\rceil \quad (32)$$

where C represents the size of the pre-sorted wave array. The visualization result of mapped indices is shown in Figure 4b. It can be seen that as the z value increases, the filtering interval continues to increase, which means that we can directly calculate the array interval instead of judging if $|\lambda\omega| \geq kL$ in Algorithm 3.

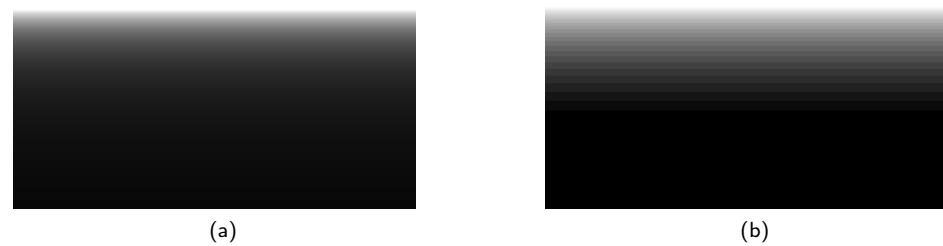


Figure 4. The visualization result of the directly calculated index where wave counts $C = 30$. The gray scale in $[0, 1]$ times C represents different filtering interval. As L increases, a large number of waves are filtered out. (a) visualization of threshold L . (b) visualization of mapped index by Equation (32).

4. Results

We executed our system, developed in C++ and GLSL 4.5, on an Intel Core i5-12400F processor (Intel, Santa Clara, CA, USA) with 16 GB of memory (six cores) and an NVIDIA GeForce RTX 2060 GPU (NVIDIA, Santa Clara, CA, USA). Within this system, our core algorithms are integrated, and we will present our key findings in the subsequent sections.

To ensure a fair comparison, we exclusively focus on the time required for wave modeling and analytical normal calculation, both essential for shading purposes. Complex shading processes like atmospheric scattering or subsurface scattering are not taken into account in our evaluation.

With a 1280×800 resolution and a camera height of 6 m, two quad patches, and maximum tessellation level $N = 64$, using a NVIDIA Nsight Graphics tool to capture the frame, we obtain ≥ 1500 fps (disable v-sync) with 60 waves with $\lambda \in (0.01, 30)$ (1500 fps contains the time cost brought by the context of GLFW window form library and imgui GUI library). As described in Section 3.4, we analytically calculate normals per pixel for shading. Since a large number of high-frequency waves are filtered out by low-pass filters in the modeling process, this gives a 0.184 ms time cost of the superposition of waves totally per frame, and when the camera height rises to 60 m, the total time cost of modeling and analytic normal calculating has been shrunk to 0.152 ms, and if we remove our self-adaptive filtering model, the time cost rises to close to 0.412 ms, no matter what camera height is selected, which proves our LOD strategy is effective. To verify the effectiveness of our self-adaptive filtering method and show its balance between performance and speed, we compare our algorithm with three current solutions: the built-in ocean water system of the Unreal 5 Engine based on Gerstner waves and a tile-based LOD system; NVIDIA WaveWorks 2.0 based on FFT and a Continuous Distance-Dependent LOD system; and a Quadtree tessellation LOD system.

To ensure the utmost fairness, after several tests, we used 100 waves in the Gerstner-wave-based method (ours, Unreal 5, Quadtree tessellation). As for the FFT-based method, known for its remarkable capabilities in simulating large-scale waveforms, WaveWorks 2.0, we adopted the configuration from NVIDIA's official samples. The comparative outcomes are presented in Table 1.

Figure 5 and Table 1 present the visualization outcomes of four distinct methods. Within Figure 5b, it becomes apparent that NVIDIA WaveWorks 2.0 stands out in terms of visual fidelity. Nonetheless, this superiority is offset by significant drawbacks, including elevated computation times on general-purpose hardware and intricate challenges in engineering execution, which collectively hinder its practical deployment. Furthermore, it necessitates additional multi-resolution FFT computations to fulfill anti-aliasing prerequisites, thereby exacerbating the IO load. As a result, FFT-based approaches find their niche primarily in high-end console or PC gaming environments, as well as in the realm of cinematic visual effects. Meanwhile, their application might not extend seamlessly to more resource-constrained platforms such as web applications and mobile games, or to specific use cases within ocean engineering, including navigation systems [1] and marine simulation software [2]. These considerations do not align with our present objectives.

Table 1. Average time (ms) comparison among our self-adaptive filtering work, Unreal 5 ocean system, NVIDIA WaveWorks 2.0, and Quadtree tessellation. Resolution is set to 1280×800 and all data come from NVIDIA Nsight Graphics and Unreal 5 built-in performance profiler.

Method	Time Cost (ms)	FPS	FPS (Disable v-Sync)
Ours	0.553	165	≥ 1328
WaveWorks 2.0	3.872	165	≥ 563
Unreal 5 ocean system	9.542	89	≥ 85
Quadtree tessellation	6.147	165	≥ 328

In the Quadtree experiment, we utilized the traditional camera-position-based tessellation strategy for Level of Detail (LOD) purposes and calculated the normals analytically without applying any filtering to the model. The results are depicted in Figure 5c. It is evident that as $z \rightarrow \infty$, pixels near the horizontal line display significant aliasing issues. Unlike screen space tessellation, the irregularity and lack of uniformity in Quadtree nodes pose challenges in designing an effective low-pass filter. Additionally, rendering the ocean surface controlled by Quadtree nodes necessitates additional draw calls, resulting in increased time costs.

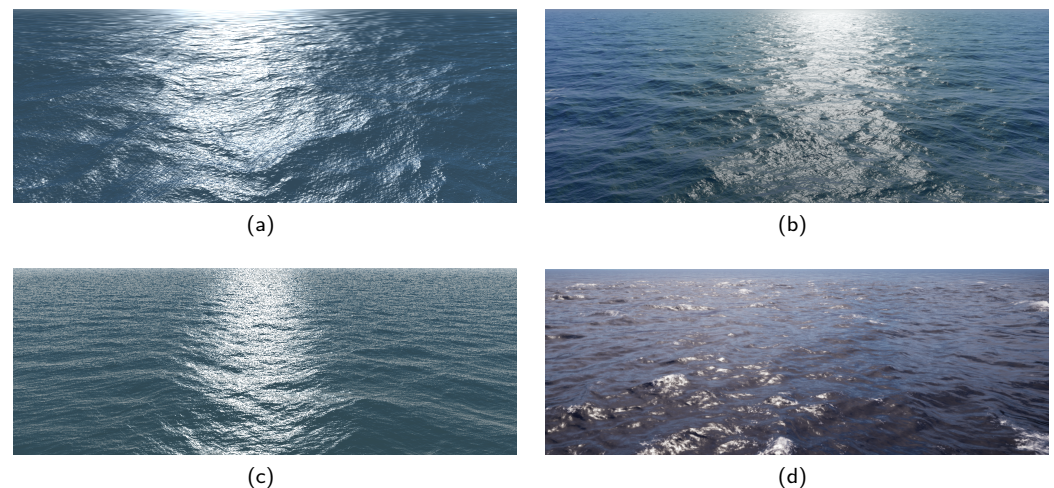


Figure 5. Rendering results of (a): our self-adaptive filtering method, (b): NVIDIA WaveWorks 2.0 based on FFT and continuous distance-dependent level of detail system, (c): Quadtree tessellation, (d): built-in ocean water system of Unreal Engine 5.

To corroborate the effectiveness of our method, we performed a series of tests aimed at assessing the rendering time required for a single frame across various camera heights $y \in (6, 66)$ within a designated scene. In the interest of ensuring fairness, we contrasted these results with the time required to activate wind waves in WaveWorks 2.0, since the shading algorithm utilized by WaveWorks 2.0 demands a substantially higher computational effort in contrast to our approach. The results, depicted in Figure 6, unequivocally demonstrate a decrease in rendering time cost as the camera height y increases. This observation unequivocally underscores the efficiency of our Level of Detail (LOD) technique.

The results from the Unreal 5 experiment, illustrated in Figure 5d, demonstrate a traditional ocean rendering pipeline that opts for interpolating geometry information within the fragment shader instead of utilizing analytic calculations. The grid near the horizon results in a discontinuous patchwork effect. Furthermore, this method capitalizes on high-quality textures crafted by artists to enhance visual expression. As a result, the specular component of the rendered ocean might not achieve the same level of realism when compared to alternative methodologies.

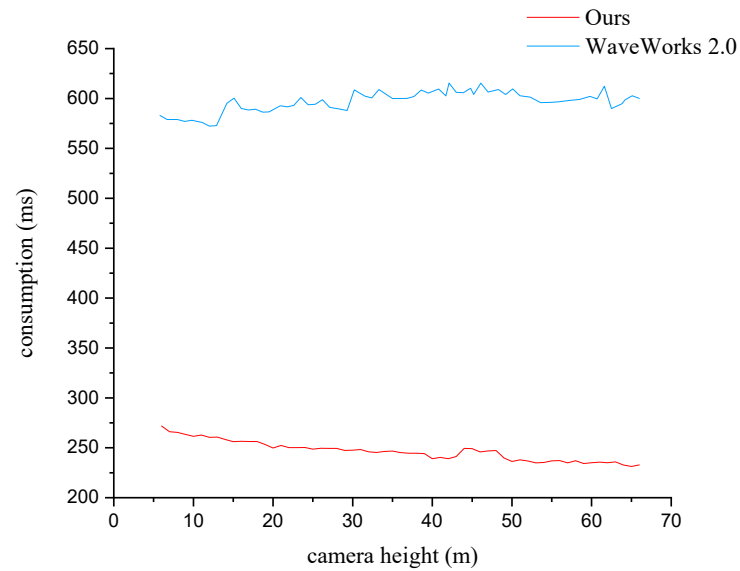


Figure 6. Time cost per frame of our method and WaveWorks 2.0 under different camera heights.

To substantiate the enhancement in the degrees of freedom for camera movement, we further selected camera viewpoints from varied perspectives for rendering. The outcomes derived are depicted in Figure 7. Upon substituting V with V' as described in Algorithm 2, the issue arising from improper handling of roll components, as illustrated in Figure 8 and further confirmed by the depiction in Figure 2b, validates that our approach indeed amplifies the camera movement's degrees of freedom without detriment to the screen space algorithm.

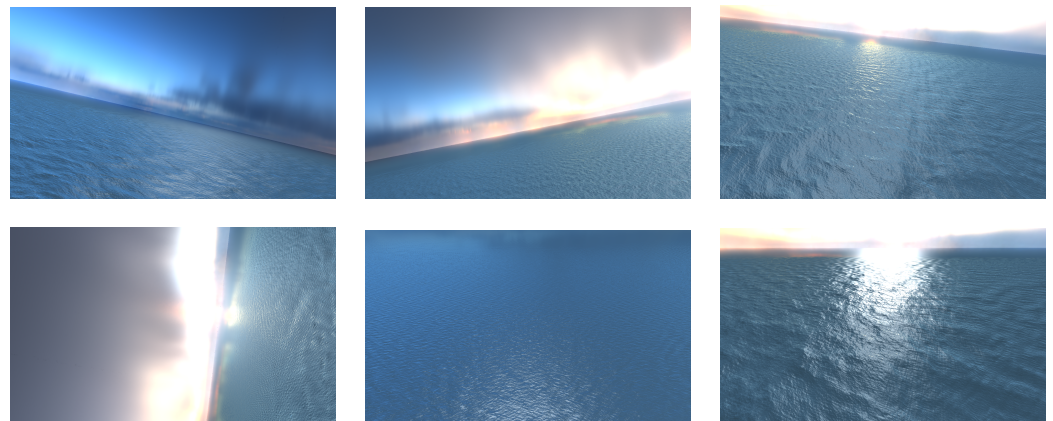


Figure 7. Result under different camera views.

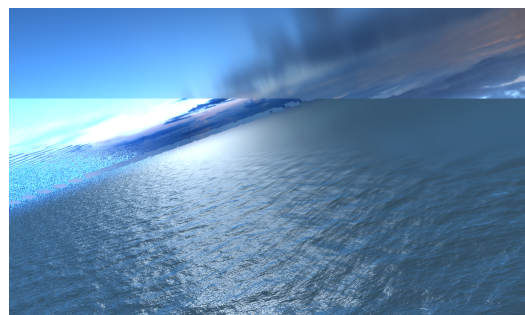


Figure 8. After replacing V with V' in Algorithm 2, the result caused by incorrect handling of roll components.

To evaluate the effectiveness of our method, we designed a questionnaire based on the System Usability Scale (SUS), encompassing the 10 questions presented in Table 2. The ratings for each questionnaire range from 1 to 5. We invited 10 experts from relevant fields to interact with our system and complete the questionnaire. The scoring procedure for each question involves a transformation of the participant's responses: for questions A, C, E, and G, the original score is reduced by 1, while for questions B, D, F, H, and J, it is subtracted from 5. These adjusted scores are then aggregated and multiplied by 2.5, converting the initial range of 0–40 to a standardized scale of 0–100. Based on research, a SUS score above 68 would be considered above average and anything below 68 is below average; however, the best way to interpret the scores is to produce a percentile ranking. The outcomes derived from the questionnaire are depicted in Figure 9.

Although there are some disadvantages in appearance due to the use of a relatively simple shading algorithm, the results presented in Table 1, Figure 5, and Table 2 do demonstrate that our approach outperforms the existing methods in terms of modeling efficiency. This superior performance can be attributed to the inherent benefits of utilizing screen space level of detail, where the granularity of primitives is determined by the screen resolution rather than the complexity of the scene itself. Concurrently, our novel self-adaptive filtering strategy adeptly addresses the challenge of non-uniform sampling across the projected grid. Moreover, in comparison to the FFT-based method, our approach of employing analytic calculations enables us to achieve competitive shading outcomes in simulations with a significantly reduced wave count.

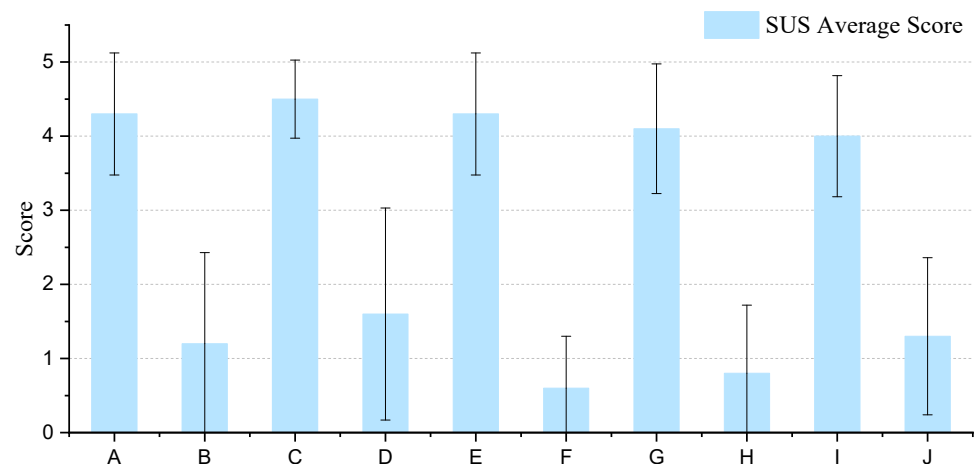


Figure 9. Score of System usability scale questionnaire. Question items A to J are from Table 2.

Table 2. System usability scale questionnaire design and the average score.

Qusetion		Average Score
A	I think that I would like to use this system frequently.	4.3
B	I found the system unnecessarily complex.	1.2
C	I thought the system was easy to use.	4.5
D	I think that I would need the support of a technical person to be able to use this system.	1.6
E	I found the various functions in this system were well integrated.	4.3
F	I thought there was too much inconsistency in this system.	0.6
G	I would imagine that most people would learn to use this system very quickly.	4.1
H	I found the system very cumbersome to use.	0.8
I	I felt very confident using the system.	4
J	I needed to learn a lot of things before I could get going with this system.	1.3
Final Score		89.25

5. Conclusions

This article introduces an enhanced screen space level of detail strategy, which is innovatively based on the camera view, coupled with an adaptive filtering strategy based on virtual camera poses. The combination of these strategies results in a detailed open sea surface rendering solution that is easy to deploy without the necessity for specialized hardware extensions. By enhancing the motion degrees of freedom of virtual cameras while also improving performance and appearance, this approach represents a significant advancement in the field of ocean engineering. For example, in aircraft [32,33] or marine simulators [2], our method can effectively overcome the lack of support for camera rolling in conventional screen space strategies while simultaneously achieving the efficiency of screen space strategy, which provides a new solution for naval applications. Drawing inspiration from NVIDIA's Variable Rate Shading (VRS) technique [34]—which intelligently applies varying shading rates across different regions—we envisage integrating our method with VRS in future works. We aspire to leverage our method within the virtual reality domain, aiming to propel the frontier of technological advancements further.

Author Contributions: This study is the result of collaborative teamwork. Conceptualization, X.W.; Investigation, X.D.; Methodology, X.D.; Software, X.D.; Validation, X.D.; Writing—original draft preparation, X.D. and J.L.; Writing—review and editing, X.D. and X.W.; Funding acquisition, X.W.; supervision, X.W. All authors have read and agreed to the published version of the manuscript.

Funding: Xinjie Wang was supported by the National Key R&D Program of China (# 2022YFC2803805), the Fundamental Research Funds for the Central Universities (# 202313035), and the Shandong Provincial Natural Science Foundation of China (# ZR2021QF124).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Li, B.; Wang, C.; Li, Z.; Chen, Y. A practical method for real-time ocean simulation. In Proceedings of the 2009 4th International Conference on Computer Science & Education, Nanning, China, 25–28 July 2009; pp. 742–747. [CrossRef]
- Ren, H.; Jin, Y.; Chen, L. Real-time rendering of ocean in marine simulator. In Proceedings of the 2008 Asia Simulation Conference—7th International Conference on System Simulation and Scientific Computing, Beijing, China, 10–12 October 2008; pp. 1133–1136. [CrossRef]
- Tessendorf, J. Simulating ocean water. *Simulating Nat. Realistic Interact. Tech. SIGGRAPH* **2004**, *1*, 5. Available online: <https://api.semanticscholar.org/CorpusID:1674789> (accessed on 21 March 2024).
- Abrashkin, A.; Pelinovsky, E. Gerstner waves and their generalizations in hydrodynamics and geophysics. *Uspekhi Fiz. Nauk* **2021**, *192*, 491–506. [CrossRef]
- Blinn, J.F. Simulation of wrinkled surfaces. *SIGGRAPH Comput. Graph.* **1978**, *12*, 286–292. [CrossRef]
- Bruneton, E.; Neyret, F.; Holzschuch, N. Real-time Realistic Ocean Lighting using Seamless Transitions from Geometry to BRDF. In *Computer Graphics Forum*; Blackwell Publishing Ltd.: Oxford, UK, 2010; Volume 29, pp. 487–496. [CrossRef]
- Gerstner, F. Theorie der Wellen. *Ann. Der Phys.* **1809**, *32*, 412–445. [CrossRef]
- Yuksel, C.; House, D.H.; Keyser, J. Wave particles. *ACM Trans. Graph.* **2007**, *26*, 99–es. [CrossRef]
- Jeschke, S.; Wojtan, C. Water wave packets. *ACM Trans. Graph.* **2017**, *36*, 1–12. [CrossRef]
- Jeschke, S.; Skřivan, T.; Müller-Fischer, M.; Chentanez, N.; Macklin, M.; Wojtan, C. Water surface wavelets. *ACM Trans. Graph.* **2018**, *37*, 1–13. [CrossRef]
- Fournier, A.; Reeves, W.T. A simple model of ocean waves. In Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA, 23–25 August 1986; Volume 20, pp. 75–84. [CrossRef]
- Kryachko, Y. Using vertex texture displacement for realistic water rendering. *GPU Gems* **2005**, *2*, 283–294.
- Bridson, R. *Fluid Simulation for Computer Graphics*; CRC Press: Boca Raton, FL, USA, 2015. [CrossRef]
- Chang, Z.; Han, F.; Sun, Z.; Gao, Z.; Wang, L. Three-dimensional dynamic sea surface modeling based on ocean wave spectrum. *Acta Oceanol. Sin.* **2021**, *40*, 38–48. [CrossRef]
- Lee, N.; Baek, N.; Ryu, K.W. *Real-Time Simulation of Surface Gravity Ocean Waves Based on the TMA Spectrum*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 122–129.

16. Ang, N.; Catling, A.; Ciardi, F.C.; Kozin, V. The technical art of sea of thieves. In Proceedings of the ACM SIGGRAPH 2018 Talks, Vancouver, BC, Canada, 12–16 August 2018; SIGGRAPH '18; Association for Computing Machinery: New York, NY, USA, 2018. [\[CrossRef\]](#)
17. Dimitrijević, A.; Rančić, D. High-performance Ellipsoidal Clipmaps. *Graph. Model.* **2023**, *130*, 101209. [\[CrossRef\]](#)
18. Schütz, M.; Krösl, K.; Wimmer, M. Real-Time Continuous Level of Detail Rendering of Point Clouds. In Proceedings of the 2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR), Osaka, Japan, 23–27 March 2019; pp. 103–110. [\[CrossRef\]](#)
19. Fu, H.; Yang, H.; Chen, C.; Zhang, H.; Hao, M.; Yu, T. Adaptive LOD representation of terrain model based on quad-tree. In Proceedings of the International Conference on Signal Image Processing and Communication (ICSIPC 2021), Chengdu, China, 16–18 April 2021; Volume 11848, pp. 258–264.
20. van der Laan, W.J.; Green, S.; Sainz, M. Screen space fluid rendering with curvature flow. In Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, Boston, MA, USA, 27 February–1 March 2009; I3D '09; pp. 91–98. [\[CrossRef\]](#)
21. Yang, X.; Pi, X.; Zeng, L.; Li, S. GPU-based real-time simulation and rendering of unbounded ocean surface. In Proceedings of the Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05), Hong Kong, China, 7–10 December 2005. [\[CrossRef\]](#)
22. Zhang, X.; Zhang, P.; Ru, Y.; Liu, C.; Qian, B.; Fu, Y.; Wang, B.; Zhang, S. Design and Practice of Virtual Simulation Teaching Experimental Platform for Marine Oil and Gas Exploration. *Exp. Sci. Technol.* **2021**, *19*, 54–59.
23. Li, R.; Sun, T.; Li, G. Application of three-dimensional GIS to water resources. In Proceedings of the 2011 19th International Conference on Geoinformatics, Shanghai, China, 24–26 June 2011; pp. 1–4. [\[CrossRef\]](#)
24. Weerasinghe, M.; Sandaruwan, D.; Keppitiyagama, C.; Kodikara, N. A novel approach to simulate wind-driven ocean waves in the deep ocean. In Proceedings of the 2013 International Conference on Advances in ICT for Emerging Regions (ICTer), Colombo, Sri Lanka, 11–15 December 2013; pp. 28–37. [\[CrossRef\]](#)
25. Post, F.H.; van Walsum, T. Fluid Flow Visualization. In *Focus on Scientific Visualization*; Hagen, H., Müller, H., Nielson, G.M., Eds.; Springer: Berlin/Heidelberg, Germany, 1993; pp. 1–40. [\[CrossRef\]](#)
26. An, W.; Li, J.; Zhao, Y.; Chen, H.; Wang, Y.; Su, T. R&D of Underwear Oil Spill Numerical Simulation and 3D Visualization System in Deepwater Area. *Aquat. Procedia* **2015**, *3*, 165–172. [\[CrossRef\]](#)
27. Pierson, W.J., Jr.; Moskowitz, L. A proposed spectral form for fully developed wind seas based on the similarity theory of S. A. Kitaigorodskii. *J. Geophys. Res.* **1964**, *69*, 5181–5190. [\[CrossRef\]](#)
28. Washington University in St. Louis. (n.d.). Airy Wave Theory. Available online: <https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.5/books/stm/default.htm?startat=ch06s02ath143.html> (accessed on 21 March 2024).
29. Louis, S.; Lapierre, L.; Onmek, Y.; Dejean, K.G.; Claverie, T.; Villéger, S. Quaternion based control for robotic observation of marine diversity. In Proceedings of the OCEANS 2017, Aberdeen, UK, 19–22 June 2017; pp. 1–7. [\[CrossRef\]](#)
30. Fréchet, J. Realistic simulation of ocean surface using wave spectra. In Proceedings of the First International Conference on Computer Graphics Theory and Applications (GRAPP 2006), Setúbal, Portugal, 25–28 February 2006; pp. 76–83.
31. Laine, S.; Karras, T.; Aila, T. Megakernels considered harmful: Wavefront path tracing on GPUs. In Proceedings of the 5th High-Performance Graphics Conference, Anaheim, CA, USA, 19–21 July 2013; HPG '13; pp. 137–143. [\[CrossRef\]](#)
32. Ratzel, M. Concept Study for the Use of a Render Engine for Computer Games in a Flight Simulation. Doctoral Dissertation, DLR-Institut für Flugsystemtechnik, Braunschweig, Germany, 2020.
33. Wang, G.; Tan, S.; Song, G.; Wang, S. Amplitude and Phase Computable Ocean Wave Real-Time Modeling with GPU Acceleration. *J. Mar. Sci. Eng.* **2022**, *10*, 1208. [\[CrossRef\]](#)
34. Rolff, T.; Schmidt, S.; Li, K.; Steinicke, F.; Frintrap, S. VRS-NeRF: Accelerating Neural Radiance Field Rendering with Variable Rate Shading. In Proceedings of the 2023 IEEE International Symposium on Mixed and Augmented Reality (ISMAR), Los Alamitos, CA, USA, 16–20 October 2023; pp. 243–252. [\[CrossRef\]](#)

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.