INTERACTIVE WATER SURFACES USING GPU BASED eWAVE ALGORITHM IN A GAME PRODUCTION ENVIRONMENT

A Thesis

by

SOUMITRA GOSWAMI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Dr. Francis Quek
Committee Member,	Dr. Ergun Akleman
Committee Member,	Dr. John Keyser
Advisory Member,	Dr. Jerry Tessendorf
Head of Department,	Tim McLaughlin

November 2019

Major Subject: Visualization

ABSTRACT

Past two years we have seen a myriad of games set in the ocean and around water. Fluids are becoming a more substantial part of video games with larger production budgets. In the past decade, we have seen a shift in real-time fluids from being a static element to an interactive entity. Even though game computation budgets are increasing, the amount given to fluid interaction and other forms of visual effects is still relatively minuscule. This budget creates a need for methods that are two to three times faster than real-time to be useful in production pipelines. GPU pipelines allow for this kind of fast computation at a fraction of the cost.

This thesis implements an algorithm that is highly parallelizable and relatively more accurate as it takes into consideration the dispersion term of the wave equation. We compare the results with the current best technique proposed by Jeschke et al. [2018]. The method implemented is a height-field based approach which involves running the wave equations in the Fourier Domain (Wave frequency domain by running a Fourier transform on the height field), solving the linearized wave equation as well as the simplified dispersion term in this domain and then transforming the result back into the real space height data. We implement this algorithm in the Unreal Engine 4, which is a popular commercial game engine used to develop big-budget games. Tests are performed using the GPU profiler, provided by the engine, to prove the algorithm's viability in a production environment.

CONTRIBUTORS AND FUNDING SOURCES

This work was supervised by a thesis committee consisting of Dr.Francis Quek and Dr.Ergun Akleman of the Department of Visualization and Dr.John Keyser of the Department of Computer Science. Dr. Jerry Tessendorf of Department of Computing, Clemson University, is acting as an Advisory Member. All work for the thesis was completed by the student, under the advising of Dr.Francis Quek of the Department of Visualization.

There are no outside funding contributions to acknowledge related to the research and compilation of this document.

NOMENCLATURE

2D,3D	Two-Dimensional, Three-Dimensional
BE	Bernoulli's Equation
CPU	Central Processing Unit
FFT	Fast Fourier Transformation
GPU	Graphical Processing Unit
NSE	Navier-Stokes Equation
UE4	Unreal Engine 4

1. INTRODUCTION

1.1. Background

The interaction of water in games is crucial to instill a feeling of immersion and believability in the world. Current games have not been shy in developing large open worlds with large amounts of water. People have an intuitive feel of how water moves and interacts with the world around us. For a few decades, there was not enough processing power to make water surfaces look realistic, far from being interactive. Up until recently, interaction with water surfaces has been a hard subject to tackle and often overlooked to give way to other essential tasks the game needs to perform. The processing power of Graphics Processing Units(GPUs) today has allowed for video game budgets to include increasing quality of water interaction.

Fluid surfaces, like water, are tough to render compared to other objects. The reason for this is that water has specific properties which make it hard to mimic accurately:

1. It is incredibly deformable and dynamic. The deformations caused by interactions are incredibly complex, which is computationally heavy to compute accurately. Complexity leads to approximation, which profoundly affects the overall look of water.
2. The look of the water poses another computational challenge, namely reflection and refraction of light. The look also changes based on the view angle of the viewer from the surface of the water. This effect known as the fresnel effect is seen in Figure 1.1 (b).

3. Finally, it is a physical phenomenon. We as humans deal with water every day. The feel of water is innately ingrained in us. So the simplification of water becomes noticeable to the average viewer. However, at the same time, if done right, it tends to draw the viewer.

Figure 1.1: (a) Photograph of complicated water behavior (b) Photograph showcasing the fresnel effect of how behaviour changes from refraction to reflection as the view angle changes.

As explained, water effects can be extremely complicated to compute but not impossible. The use of water is very prevalent in movies today. Almost all visual effects(VFX) heavy movies have some form of fluid dynamics. How is this possible, if water is so complicated and computationally heavy? Movies, advertisements, and other forms of non-interactive media work on a different pipeline from realtime media. The implication is such that non-realtime media can split its computation resources amongst multiple computers and artists, without the limitation of framerate(time taken to generate each frame). In the case of realtime media such as game production, the computation of the entire pipeline - including gameplay, Artificial Intelligence(AI), Rendering and special effects - needs to be done in one computer with the

added constraint of running at 60 frames per second(fps). As a comparison, a frame of the ocean from Life of Pi may take several hours to generate, while in games given the production budgets the same ocean would need to be done in under ten milliseconds.

(a) (b)

Figure 2.1: (a) Ocean seen in the movie Life of Pi (2012) and (b) Ocean seen in the game Assassin's Creed Odyssey (2019)

So we are faced with a task to make water look believable at a fraction of the cost. Game engines have created a pipeline, where game programming and artificial intelligence (AI) tasks are sent to the CPU, while GPU handles anything to do with rendering, including Lighting, Shading, Instantiating, and Special Effects. This pipeline split is partly because of the rapid growth of GPU innovation over the years. GPU architecture allows for rapid texture computation and high amounts of parallelization as compared with CPUs. Hence, we can leverage the power of the GPU to compute water computations quickly and efficiently by making sure the algorithms are simple instructions that are parallelizable. Hence, allowing texture-based approaches to be computed at a much faster rate and be feasible in game production.

One of the essential simplifications of the texture-based approach is using heightfields. In this method, the water surface is represented simply by height values on a Two Dimensional (2D) grid. This technique reduces simulation computation from Three Dimensional (3D) cell to a 2D grid. The main drawback of these methods is the loss of 3D details, such as splashes, but there are methods to add these back when necessary. This technique has been seen in use in the simulation of ocean surfaces in games like the Assassin's Creed franchise, Sea of Thieves, and many others[Ang et al. 2018; Torres and Mike 2012]. The technique has added much realism to the scenes and emboldened Game Developers to tackle ocean surfaces in their games. They are thus expanding the capabilities of what is possible in real-time.

Interactivity with the player is another significant aspect to discuss. So far simulation of ambient water - including breaking of waves and ocean surfaces - does not consider the involvement of a player. In games, players interact approximately with elements as rigid bodies. Hence, when dealing with physical interaction, it becomes essential to model rigid body-water coupling. With players interacting with water will be analogous to rigid body interaction with water. Rigid bodies are objects which do not deform from any external force. Rigid bodies also calculate forces on their center of mass. To consider the rigid body interaction with water, we break it into two problem sets. One set will examine the force applied by rigid bodies on water to generate the waves on the surface, and the other set will examine the force applied by water onto the rigid body in regards to linear forces like buoyancy.

Several techniques provide interaction with water surfaces. There are three principal techniques utilized in current games. This thesis will introduce the eWave technique, which could be useful in games and compare it with one of the existing techniques in a game

production environment. This thesis will use the Unreal Engine 4 (UE4) as its test-bed. UE4 is a popular game engine used by many game production studios. Testing the algorithm in UE4 should provide proof of the viability of the technique in a game production environment.

1.2. Research Questions and Methodologies

This thesis strives to find a way to achieve richer detail in the water interaction while adding minimal computation time. As discussed earlier, GPUs are getting more powerful and accessible, so there is a budget for player-water interaction. There is room for implementing newer algorithms, to enhance the richness in detail of water interaction. A more detailed water simulation provides a deeper sense of reality for the player. This deeper sense is required to maintain the feeling of immersion in a player. The following main research question (RQ) arises from this goal:

> **RQ1** How does eWave, as situated in limited computational game environments, compared with existing methods as used in existing game production pipelines?

[RQ1] can be answered with simple comparisons between the different techniques using computational time and framerate as a reference. More details of the methodology are presented in Section 3 of this thesis.

1.3. Summary of Main Contributions

To answer [RQ1] the thesis details an implementation of the eWave algorithm through the GPU via the UE4 platform. Noting the limitations of the existing eWave algorithm, the contribution to the approach is the inclusion of forces produced by the water onto the geometry. This is done by calculating the buoyancy and the surface force applied to the geometry.

To evaluate the viability of both the 'vanilla' (Dr. Tessendorf's original algorithm) and 'enhanced' algorithms(my addition to the algorithm), comparisons will be made with another popular technique currently in use. These comparisons will be based on the measurement of the computation time and frame rate.

1.4 Goals of the thesis

The goal of this thesis is to study the viability of the 'enhanced' model of Tessendorf's algorithm compared to the current best technique presented by Jaschke et al[2018]. A parallel CPU prototype of both techniques will be built and compared. An Unreal Engine 4 (UE4) GPU plugin will be built incorporating the enhanced model if Tessendorf's algorithm. Here the result will be compared to the GPU prototype provided by the Jeschke et al[2018]. Comparisons will be made in terms of frame-rate and computation time. Results will be presented in detail along with performance analysis. A discussion will be presented to improve the simulation and prepare it for the production pipeline.

2. RELATED WORKS IN WATER SIMULATION

Water or Fluid interaction and simulation have been studied quite extensively in the past in the field of Physics and Engineering.

2.1) Surface waves

Early work in computer graphics focused on direct modeling of the water surface and simulation through the animation of surface waves. These methods tried to approximate physical characteristics through noise functions, trigonometrical formulations or Fourier Synthesis.

The approach to modeling ocean waves involved applying simplifications to the Navier-Stokes equations (NSE) or geometrical formulations to end up generating some form of sinusoidal waves. One of the first mathematical attempts at modeling wave surfaces in computer graphics was from Schachter [1980]. He presented a method to combine just 2-3 long-crested narrow-band waveforms as compared to the use of a large number of sinusoids used at a time. Fournier and Reeves [1986], studied oceans further and utilized Gerstner's parametric model of wave generation. This modeled waves by having particles of water take circular or elliptical stationary orbits. This method produced more believable looking waves for its time and allowed for the phenomenon of breaking waves to be modeled as well. Peachy [1986] also provided another parametric model to simulate waves approaching and breaking on a sloping beach. Peachy introduced a depth parameter to the ocean models which allowed for varying levels of depth, helping in wave modeling of a shoreline. Gonzato and Le Saec [2000] modify this model and allow for the curling of the wave crest (Breaking waves).

Ts'o and Barsky [1987] proposed a different method, called wave tracing. This involving casting rays from the skyline onto a grid to generate a spline surface. Since this method was based on linear splines to describe a surface, there was a lack of surface details. Gonzato and Le Saec [2000] addressed this problem by sampling new rays in the undersampled area. This approach of sampling was also used by Gamito and Musgrave [2002] to extract a phase map used in the parametric model, which can then be animated easily.

Mastin et al. [1987] used a spectrum-based approach. The paper used an empirical spectral model of real sea waves to generate water surfaces. Tessendorf [2004] too had his oceans waves based on Fourier Synthesis. These methods although generated realistic waves, were static (they did not handle interaction with obstacles) and assumed periodic boundaries (object passing through one side of the boundary, re-appears on the opposite side with the same velocity).

2.2 Heightmap and 2D waves

Heightmap-based solutions started to be used more to solve a lot of shortcomings of these strict modeling approaches. A heightmap is a 2D image where each pixel on the image represents the height value of the corresponding 3D grid. Heightmap methods proved to be faster to simulate, lighter on memory and allowed for solid-liquid interaction. 2D Shallow Water Equations(SWE) which are essentially a simplification of Navier Stokes Equations by assuming constant pressure and a velocity gradient along the vertical axis. Kass and Miller [1990] used finite-differences to discretize the differential SWE. They assume the volume of water to be a height grid of virtual columns of water. O'Brien and Hodgins [1995] further improved this

method using virtual pipes. The water flows between these virtual columns of water through a difference in hydrostatic pressure. They are able to calculate the force generated from changing heights to be able to generate particle splashes.

Wang et al. [2007] improved upon the SWE by adding gravity and surface tension effects. Kallin [2008] simulated SWE with an adaptive grid to the simulations allowing use in large areas of water in open-world games. Thurey et. al [2007] used SWE to model real-time overturning of waves (breaking). Chentanez and Muller [2010], combined heightmap based Shallow Water surface waves with particle-based breaking and splashes. Chentanez also added a method to add small waves advected with the water flow. Ojeda and Susin [2013] utilized a 2D Lattice Boltzmann Method[LBM] simulator to solve SWE in real-time on top of a heightfield terrain.

SWEs are fast and efficient, but they lack the dispersion term (speed of the wave). So these methods produce waves that move at the same speed. Yu et al. [2012] used 3D particle-based simulation to generate a triangular mesh at the surface of the liquid. Yu et al. then uses a Bi-Laplacian surface energy formulation to generate surface waves. This allowed for incorporating the dispersion term.

Tessendorf [2004; 2014] incorporates the dispersion term and formulated his method by discretizing a linearized Bernoulli equation and an approach to remove periodicity. Other researchers utilized other convolution-based approaches to achieve correct dispersion speeds [Ottosson 2011; Loviscach 2002]. The practical problem of convolution kernel sizes taking up the entire simulation region occurs with these methods. Canabal et al. [2016] solved this problem by using pyramid filters and shadowed convolution operations.

We will be solving the convolution kernel problem by removing the convolution aspect of Tessendorf [2014] algorithm, but instead utilizing the FFT algorithm also presented in the note. We will be removing periodicity caused by the Fourier domain by dampening the wave at the boundaries.

Yuksel et. al [2007a] developed a hybrid approach to wave simulation. He proposed wave particles that represent the local wave crest moving with a calculated speed. Jeschke and Wojtan [2017] expanded on this and introduced wave packets. Each packet moves with a group speed and contains a group of waves traveling at a calculated phase speed. Jeschke et al. [2018] further expanded on this by providing a new theoretical model that transports amplitude information as a function of space, frequency and wave direction combined on slowly modulating waves. They formulated a new method to discretize this complex amplitude form. This formulation allows for the real-time propagation of the waves. I will be comparing an improved Tessendorf [2014] algorithm with Jeschke's [2018] model to study the viability of both methods.

2.3 Water in video games

Simulating water is a highly computational process and video games are only allowed a small time-frame for physics simulations in a total time of the game loop. So most of the techniques (barring a few) presented above are too expensive to be done in real-time. Video Games for ages have utilized clever simplifications and tricks to showcase a complicated effect. Water has been a part of games for a long time. In 2D games, water has been represented by

simple animations. The interaction involved predefined artist animated textures spawned on location[See Sonic figure].

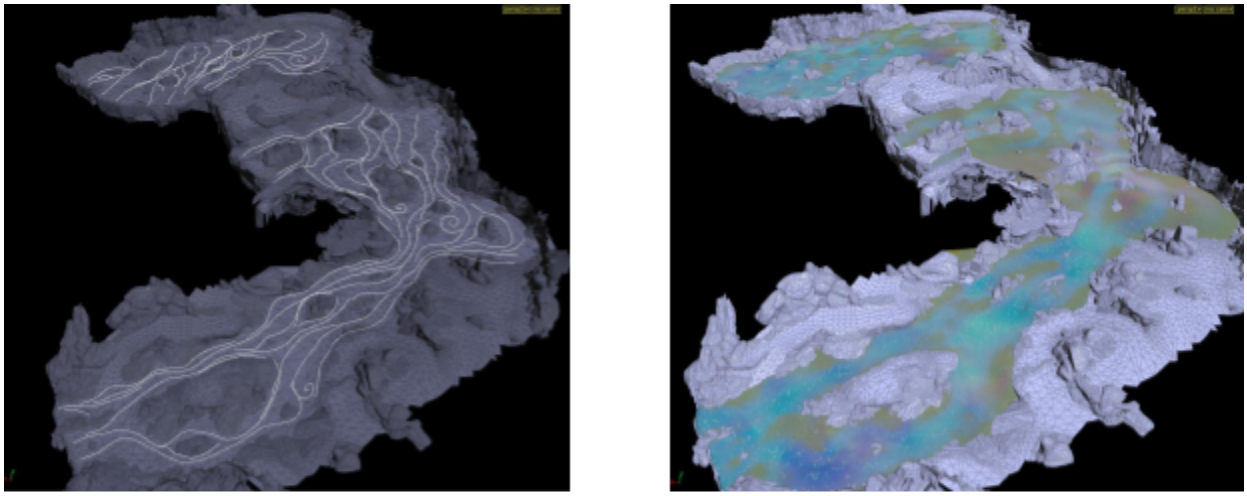
(a) (b)

Figure 2.1: (a) Animated textures seen with Sonic the Hedgehog (1991) and (b) Projected 2D noise on a 3D mesh seen in Doom 2 (1994)

3D games applied a similar concept, a static mesh was used with an animated noise texture - a flat 2D image projected onto a 3D mesh. Doom 2 [1994] called this technique an "animated flat". Max et al. [1993] introduced a method of visualizing movement through the use of flow maps. These are artistically made splines that drive texture movements. This technique is used in animated rivers, streams, etc. Uncharted Franchise has utilized this technique for its rivers [Brinck et al. 2016; Ochoa and Holder 2012; Dog 2016] [SideFX, FX Adventures, Figure below].

In the past decade, we are seeing a lot more physically based approximations done in games. Shallow Water Equations(SWE) have been utilized in solid-liquid interaction in recent games like Assassin's Creed [Torres and Mike 2012]. The use of a variation of Brian and Hodgins' [1995] pipe method is utilized and seen in Cities: Skylines(2015) [Kellomäki 2017].

We are seeing the use of full 3D particles in games like Borderlands 2 (2012) as well. They utilized the Nvidia PhysX library in the PC version to generate their 3D particles. Battlefield One (2017) utilized animated meshes using Vector Maps to generate some of their splash effects in their game. However, these methods still aren't used at a large scale because of the heavier computational requirements. So, in most cases, we see the use of 3D sprites (camera facing textures) that have a pre-rendered animation playing on them to generate the illusion of a larger, more complicated effect. [Glad 2017]



(a) (b)

Figure 2.2: (a) The flow curves seen which define the flow of the river. (b) the resultant water with textures flowing based on the flow maps.

Uncharted(2007-2016) has also utilized a variation of Yuksel et al.'s [2007b] wave particles to generate their oceans and most other water interactions. They combine Grestner [Fournier and Reeves 1986] parametric waves with artistically generated wave crests to create their large ocean scenes [Ochoa and Holder 2012]. Recent games like Sea of Thieves(2017) and Assassin's Creed have also utilized Tessendorf [2004] FFT models to generate their oceans.[Ang et al. 2018; Torres and Mike 2012].

We are yet to see non-SWE methods utilized in games. As we have discussed earlier, SWE lack the dispersion component and hence all generated waves move at the same wave speed. This paper compares two non-SWE methods and presents a higher quality water interaction without too much compromise in computational costs.

3. THEORETICAL METHODOLOGY

3.1 Bernoulli's Equations

The generation of the eWave algorithm occurs from the linearization of Bernoulli's Equation of fluid Motion, free surface motion, and mass conservation equations. This is seen in the short class course notes[Tessendorf 2008]. So, taking a look at the equations in their original form.

$$\frac{\partial}{\partial t}\phi(\mathbf{x},t) + \frac{1}{2}|\nabla\phi|^2 = -gh(\mathbf{x},t) \qquad \textit{Fluid Motion (3.1)}$$

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right)\phi = 0 \qquad \textit{Mass Conservation (3.2)}$$

$$\frac{\partial h}{\partial t} + \nabla\phi \cdot \nabla h = \frac{\partial \phi}{\partial y} \qquad \textit{Free Surface Motion (3.3)}$$

Here at position **x**(*x,y,z*) and time **t**,

ϕ- Velocity Potential of the grid ***h*** - Height-field of the grid

g- Gravity force applied

In order to linearize the equations, one can consider $|\nabla\phi|^2$ in eq (3.1) to be negligible and hence ignored from the equation. The same can be said about $\nabla\phi \cdot \nabla h$ in eq (3.3). The product can be considered negligible and also ignored for now. The ignored terms from eq (3.1) and eq (3.3) will come back later as self-advection in our simulation.

So the linearized versions of the equations now look like this,

$$\frac{\partial}{\partial t}\phi(\mathbf{x},t) = -gh(\mathbf{x},t) \tag{3.4}$$

$$\frac{\partial h}{\partial t} = \frac{\partial \phi}{\partial y} \tag{3.5}$$

We can use eq (3.2) to calculate $\frac{\partial \phi}{\partial y}$ which when inserted in eq (3.5) becomes,

$$\frac{\partial h}{\partial t} = \sqrt{-\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2}\right)}\phi$$

If we assume the column of water under each grid on the surface to have uniform pressure and density, we successfully convert a 3D problem to a 2D solution. Now that we are dealing with a 2D equation, we can see that $\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2}\right)$ is a laplacian (∇^2) in a 2D context. So finally we get

$$\frac{\partial h}{\partial t} = \sqrt{-(\nabla^2)}\phi \tag{3.6}$$

So, in the end, eq(3.4) and (3.6) become the starting point of the eWave algorithm.

3.2. Conversion to the exponential form

To simplify the equations further we need to simplify them into the exponential form as seen in [Tessendorf 2014]. But first we need to set up eq(3.4) and eq(3.6) as a couplet.

$$W(x,t) = \begin{bmatrix} h(x,t) \\ \phi(x,t) \end{bmatrix} \tag{3.7}$$

The linear equations eq(3.4) and eq(3.6) translate to our couplet having the equation of motion as below,

$$\frac{\partial W(x,t)}{\partial t} = MW(x,t) \tag{3.8}$$

Where M is the matrix,

$$M = \begin{bmatrix} 0 & \sqrt{-\nabla^2} \\ -g & 0 \end{bmatrix} \tag{3.9}$$

Eq(3.8) can be represented as an exponential solution

$$W(\boldsymbol{x},t) = \boldsymbol{e}^{Mt}\, W(\boldsymbol{x},0) \tag{3.10}$$

3.4. Solving the exponential form

Anything to a power of a Matrix is not defined in mathematics. Hence, we need to convert it to a more linear solution. This is where Taylor series comes to our aid.

Once we expand the exponential into a Taylor series we get:

$$e^{Mt} = \sum_{n=0}^{\infty} \frac{M^n t^n}{n!} \tag{3.11}$$

Since n is tending to infinity, we can split the powers into even and odd sets.

$$e^{Mt} = \sum_{n=0}^{\infty} \frac{M^{2n} t^{2n}}{(2n)!} + \sum_{n=0}^{\infty} \frac{M^{2n+1} t^{2n+1}}{(2n+1)!} \tag{3.12}$$

Now, we know our M from eq(3.9). We need to generate the identities to M^{2n} and M^{2n+1}. We can deduce the identities by working through them to get,

$$M^{2n} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \left(-g\sqrt{-\nabla^2} \right)^n \tag{3.13}$$

$$M^{2n+1} = M\left(-g\sqrt{-\nabla^2} \right)^n \tag{3.14}$$

Substituting the matrices in eq(3.12) with eq(3.13) and eq(3.14) we get,

$$e^{Mt} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \sum_{n=0}^{\infty} \frac{\left(-g\sqrt{-\nabla^2}\, t^2 \right)^n}{(2n)!} + Mt \sum_{n=0}^{\infty} \frac{\left(-g\sqrt{-\nabla^2}\, t^2 \right)^n}{(2n+1)!} \tag{3.15}$$

If we take a closer look, the series on the left is the infinite series for the cosine and the other series is the infinite series for the sine. So if we define $\widehat{\omega} \equiv \sqrt{g\sqrt{-\nabla^2}}$

$$e^{Mt} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cos(\widehat{\omega} t) + \frac{M}{\widehat{\omega}} \sin(\widehat{\omega} t) \tag{3.16}$$

This then expands in matrix form to,

$$e^{Mt} = \begin{bmatrix} \cos(\widehat{\omega} t) & \dfrac{\sqrt{-\nabla^2}}{\widehat{\omega}} \sin(\widehat{\omega} t) \\ -\dfrac{g}{\widehat{\omega}} \sin(\widehat{\omega} t) & \cos(\widehat{\omega} t) \end{bmatrix} \tag{3.17}$$

We can now fit eq(3.17) into eq(3.10) to give us,

$$\begin{bmatrix} h(x,t) \\ \phi(x,t) \end{bmatrix} = \begin{bmatrix} \cos(\widehat{\omega} t) & \dfrac{\sqrt{-\nabla^2}}{\widehat{\omega}} \sin(\widehat{\omega} t) \\ \dfrac{-g}{\widehat{\omega}} \sin(\widehat{\omega} t) & \cos(\widehat{\omega} t) \end{bmatrix} \begin{bmatrix} h(x,0) \\ \phi(x,0) \end{bmatrix} \tag{3.18}$$

Since ∇ and $\widehat{\omega}$ are both independent of t, we can generate a linear equation from eq(3.18) with incrementing time as such.

$$\begin{bmatrix} h(x,t+\Delta t) \\ \phi(x,t+\Delta t) \end{bmatrix} = \begin{bmatrix} \cos(\widehat{\omega}\Delta t) & \frac{\sqrt{-\nabla^2}}{\widehat{\omega}}\sin(\widehat{\omega}\Delta t) \\ \frac{-g}{\widehat{\omega}}\sin(\widehat{\omega}\Delta t) & \cos(\widehat{\omega}\Delta t) \end{bmatrix} \begin{bmatrix} h(x,t) \\ \phi(x,t) \end{bmatrix} \quad (3.19)$$

Converting the matrix form in eq(3.19) to a linearized form through simple matrix multiplication we get the final linearized equations in real space as follows,

$$h(x,t+\Delta t) = \cos(\widehat{\omega}\Delta t)\, h(x,t) + \frac{\sqrt{-\nabla^2}}{\widehat{\omega}}\sin(\widehat{\omega}\Delta t)\, \phi(x,t) \quad (3.20)$$

$$\phi(x,t+\Delta t) = \cos(\widehat{\omega}\Delta t)\, \phi(x,t) - \frac{g}{\widehat{\omega}}\sin(\widehat{\omega}\Delta t)\, h(x,t) \quad (3.21)$$

3.5. Dealing with the complex Laplacian term

As mentioned in the note [Tessendorf 2014], the complex way the laplacian(∇^2) is represented in the real space in eq(3.20) is tricky to solve.

One way to solve it has been done in previous iWave solutions. This method has issues seen in other convolution-based approaches in terms of resolution of simulation as reviewed in the related works section(Section 2.2). There have been workarounds to overcome this issue but these can be rather expensive to handle.

The note [Tessendorf 2014] provides an alternative method by solving the simulation in Fourier space, aka wave spectrum space. This space simplifies the appearance of the laplacian term considerably to turn into the absolute magnitude of the Fourier Vector (***k***). The $\widehat{\omega}$ term becomes a dispersion relation as well in this space. These changes are reflected below.

$$h(x) \longrightarrow \tilde{h}(k)$$

$$\phi(x) \longrightarrow \tilde{\phi}(k)$$

$$\sqrt{-\nabla^2} \longrightarrow \boldsymbol{k}$$

$$\widehat{\omega} \longrightarrow \omega(k)$$

Calculation of the dispersion relation is a simple equation as shown below,

$$\omega(k) = \sqrt{gk} \qquad \textit{eq(3.22)}$$

Hence, with these simplifications equations *3.20* and *3.21* now become,

$$\tilde{h}(k, t+\Delta t) = \cos(\omega(k)\,\Delta t)\;\tilde{h}(k,t) + \frac{k}{\omega(k)}\sin(\omega(k)\,\Delta t)\;\tilde{\phi}(k,t) \qquad \textit{eq(3.23)}$$

$$\tilde{\phi}(k,t+\Delta t) = \cos(\omega(k)\,\Delta t)\;\tilde{\phi}(k,t) - \frac{g}{\omega(k)}\sin(\omega(k)\,\Delta t)\;\tilde{h}(k,t) \qquad \textit{eq(3.24)}$$

3.6. Dealing with Periodicity of boundaries

Since we are running the calculations in Fourier Space, our simulation is resolution-independent making grid size a non-factor for simulation calculations. One major disadvantage of this technique though is the periodicity of the simulation grid boundaries. This can be highly unfavorable for most production situations. To deal with this issue, we simply dampen the heightfield such that the waves generated get killed as soon as they reach the boundary.

This dampening value (***D***) is calculated with the following equation,

$$D(x) = \begin{cases} \left(\frac{x}{L}\right)^{\alpha} & \frac{x}{L} < 1 \\ \frac{abs\left(x - L_x\right)}{L} & \frac{abs\left(x - L_x\right)}{L} < 1 \\ 1 & else \end{cases}$$

$$D = D(x) \cdot D(y)$$

$$D \in [0, 1]$$

eq(3.25)

Where,

- x,y are horizontal and vertical pixel positions respectively.
- Grid size is from x = 0 to x = L_x. First line of equation in group eq (3.25) is for any component.
- L is the size of the linear taper region.
- α is the damping factor.

Note: We get the best visual results by setting L to 10% of the corresponding grid size and $\boldsymbol{\alpha}$ set to 0.05.

The resulting Dampening value (D) lies between [0,1] and is then multiplied to both height(h) and vel potential(ϕ).

$$h = h \cdot D$$
$$\phi = \phi \cdot D \qquad \textit{eq(3.26)}$$

3.7 Static Obstruction

Static Obstructions are easily implemented with an Obstruction mask. We generate a map of where static obstructions are around the grid. Static obstructions are masks that don't move and reflect waves back.

Obstruction mask is inverted and multiplied to the heightfield and velocity potential field. Suppose $\boldsymbol{O_{ij}}$ is the gridded obstruction mask between [0,1], while h_{ij} and ϕ_{ij} are gridded heightmap and velocity potential maps respectively.

$$h' = h_{ij} \cdot O^I_{ij}$$
$$\phi' = \phi_{ij} \cdot O^I_{ij} \qquad \textit{eq(3.27)}$$

Where,

$$O^I_{ij} = 1.0 - O_{ij} \qquad \text{eq(3.28)}$$

3.8 Enhancing the algorithm

We enhance the algorithm by:

1) We add the concept of drift velocity which would control the advection of the asset.
2) We allow dynamic obstacles that generate waves based on the speed at which they are moving.
3) We generate horizontal displacement to add choppiness to the wave and give the waves a sense of direction.
4) We calculate the Jacobian of the horizontal displacement to generate a mask for foam generation.
5) We add self advection component to bring back the non-linear terms in eq 3.1 and eq. 3.3.

3.8.1. *Drift Velocity*

One can utilize existing flow map techniques [Max et al. 1993] to generate the drift velocity(U_{flow}) at a particular point of the simulation grid. We use this velocity to transport height (*h)* and velocity potential(ϕ) on the simulation grid. This transport of data on a velocity field is known as advection. The drift Velocity is provided by the user as a map

$$h(x,t) \leftarrow advect\left(h(x,t), U_{flow}(x) \cdot scale\right)$$
$$\phi(x,t) \leftarrow advect\left(\phi(x,t), U_{flow}(x) \cdot scale\right) \quad \text{eq(3.29)}$$

There are several advection schemes which can be utilized here. The simplest one utilized by most cases is the Semi-Lagrangian advection scheme (See [Bridson and Müller-Fischer 2007] discussion of this scheme). The issues with such a scheme is that it assumes linear motion and since we are dealing fluids the velocity field is not linear at all.

[Selle et al. 2007] proposed a stable scheme known as the Modified MacCormack method. This method is stable and incorporates second order advection. We utilize the Modified MacCormack advection scheme to advect our data.

3.8.2. *Dynamic Obstructions*

Dynamic Obstruction is an element that not only reflect waves but generate new waves, if the obstruction is in motion. Either the center of mass for the tracked object is provided (object center pivot projected onto the grid) or can be calculated from the provided projected mask.

Let us assume $\vec{O}_{ij}$ refers to a Dynamic Obstruction Map. All the existing fields are inverted inside the obstruction mask as seen below.

$$h_{amb}^{obs} = \vec{O}_{ij} \cdot \left(- h_{amb} \right)$$
$$h_{ij}^{obs} = \vec{O}_{ij} \cdot \left(- h_{ij} \right)$$
$$\phi_{ij}^{obs} = \vec{O}_{ij} \cdot \left(- \phi_{ij} \right) \qquad \textit{eq(3.30)}$$

Here, h_{amb} is the ambient waves that are already present in the simulation.

Calculating Magnitude

Let *R* be the point to be tracked to generate the magnitude of the source waves. We will use this point to determine whether or not the obstacle has moved between frames. The more the object moves the larger the magnitude.

To track R each frame, R_{prev} variable is used to store previous R data. We then calculate the magnitude by a simple linear calculation of magnitude based on *R* movement.

$$M(x,t) = \left(\frac{len\left(R_{cur} - R_{prev} \right)}{t} \right) \cdot scale \qquad \text{eq(3.31)}$$

Where , M(x,t) is the magnitude of movement and R_{cur} is the current position of the center of mass or anchor point. A custom scale parameter can be used for additional control.

Calculating Center of Mass

If the user doesn't provide an anchor point but asks for the point to be calculated from the provided dynamic obstruction map , then this can be done with a simple calculation each time step.

$$R(x) = \frac{1}{N \cdot M} \sum_{j=0}^{M} \sum_{i=0}^{N} \overrightarrow{O}_{ij} \cdot x_{ij} \qquad \textit{eq(3.32)}$$

Where,

N - number of horizontal Obstruction map samples.

M - number of vertical Obstruction map samples.

x - current pixel position.

This generates the center of mass each frame. This can get tedious every frame so a provided center of mass works out to be a lot faster.

Adding to the source

Once we have the calculated magnitude we then add it to the heightmap source ($S(x)$) in the appropriate timestep. We also add the reversed obstruction elements. This will make the areas inside the mask to be zero, hence acting like static Obstruction maps.

$$h(x) = h(x) + S(x) \cdot M(x) \cdot \Delta t \qquad \textit{eq(3.33)}$$

$$h'(x) = h(x) + h_{amb}(x) + h_{amb}^{obs}(x)$$
$$h(x) = h'(x) + h^{obs}(x)$$
$$\phi(x) = \phi(x) + \phi^{obs}(x) \qquad \textit{eq(3.34)}$$

If we consider the moving source as a dynamic obstruction map, the above set of equations will incorporate self-reflections as well.

3.8.3. *Horizontal Displacement or Choppiness*

Our algorithm so far generates just a heightmap. If we look at waves in real life, waves have a sense of direction. We can generate this via horizontal displacements. *[Creamer et al.*

1989] and (Tessendorf 2004) talk about horizontal displacement. In the fourier domain, the 2D displacement (*D(x)*) is calculated as,

$$D(x) = \frac{1}{L_x} \int -i \frac{k}{\mathbf{k}} \cdot h(k) \cdot e^{ik \cdot x} d^2k \qquad \textit{eq(3.35)}$$

$$D(x) = FFT^{-1}\left(-i \cdot \frac{k}{\mathbf{k}} \cdot h(k) \right) \cdot \frac{2\pi}{L_x} \qquad \textit{eq(3.36)}$$

As we can see we multiply the height in fourier space with the normalized wave vector($\frac{k}{\mathbf{k}}$) and -i. This will give us the necessary directionality to the wave. We note that since our inverse FFT normalizes the result by dividing by 2π , we compensate by multiplying by 2π to get back our equation in eq(3.35). Here, L_x is the size of the grid depending on the dimension.

We can also provide the artist with a parameter to control the amount of directionality they want to see in their wave. This parameter we label as *Choppiness* (λ). We see how we add choppiness to the position in eq(3.37).

$$x = x + \lambda \cdot D(x) \qquad \textit{eq(3.37)}$$

3.8.4. *Foam/Whitecap mask*

3.8.5. *Self Advection*

If we recall equation of free motion eq(3.3). We see that we ignored the operation $\nabla\phi \cdot \nabla h$ to generate our equations. Same can be said for the equation of fluid motion in eq(3.1). In this case, we have ignored the operation $\frac{1}{2}|\nabla \phi|^2$. If we want to bring in these operations, we need to define them first.

We know that ϕ is the velocity potential of the simulation. The divergence (∇) of the velocity potential can be considered as generated velocity field (*u)* of the area.

$$\nabla \phi \longrightarrow u \qquad \textit{eq (3.38)}$$

If we substitute that in these operations, we will see an interesting operation.

$$(u . \nabla) h \qquad \textit{eq(3.39)}$$

$$\frac{1}{2} |\nabla \phi \cdot \nabla \phi|$$

$$\frac{1}{2} (u \cdot \nabla) \phi \qquad \textit{eq(3.40)}$$

Here, $u \cdot \nabla$ operator is also known as *advection*. As we saw earlier advection can be used to transport data based on the velocity field. In our case, the heightfield (*h*) and velocity potential (ϕ) are being transported by calculated velocity. With this velocity, that we term as self advected velocity, we can utilize known methods to transport height and velocity potential. Hence we start seeing the generated waves interact with itself, simulating a physical phenomenon. Since, we are dealing with a simulation grid for our values, we will be using the Modified MacCormack advection scheme. (For further discussion on this advection scheme, see [Selle et al. 2007]).

FUTURE WORK

RQ2 Is the added fidelity to the player-water interaction noticeable to the player? If not, then how far can we simplify the method till the player notices the difference in experience?

[RQ2] falls under the behavioral science spectrum of things. This requires extensive user studies to be conducted on a varied demographic of players. [RQ2] will hence be beyond the scope of this thesis.

7. REFERENCES

Ang, N., Catling, A., Ciardi, F.C., and Kozin, V. 2018. The technical art of sea of thieves. *ACM SIGGRAPH 2018 Talks on - SIGGRAPH '18*. http://dx.doi.org/10.1145/3214745.3214820.

Bridson, R. and Müller-Fischer, M. 2007. Fluid simulation. *ACM SIGGRAPH 2007 courses on - SIGGRAPH '07*. http://dx.doi.org/10.1145/1281500.1281681.

Brinck, W., Maximov, A., and Jiang, Y. 2016. The Technical Art of Uncharted 4. *ACM SIGGRAPH 2016 Talks*, ACM, 60:1–60:1.

Canabal, J.A., Miraut, D., Thuerey, N., Kim, T., Portilla, J., and Otaduy, M.A. 2016. Dispersion Kernels for Water Wave Simulation. *ACM transactions on graphics 35*, 6, 202:1–202:10.

Chentanez, N. and Müller, M. 2010. Real-time Simulation of Large Bodies of Water with Small Scale Details. *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 197–206.

Creamer, D.B., Henyey, F., Schult, R., and Wright, J. 1989. Improved linear representation of ocean surface waves. *Journal of fluid mechanics 205*, 135–161.

Dog, N. 2016. FX Adventures in Uncharted 4: A Thief's End | SideFX. *sideFX.com*. https://www.sidefx.com/community/fx-adventures-in-uncharted-4-a-thiefs-end/.

Fournier, A. and Reeves, W.T. 1986. A Simple Model of Ocean Waves. *SIGGRAPH Comput. Graph. 20*, 4, 75–84.

Gamito, M.N. and Musgrave, F.K. 2002. An accurate model of wave refraction over shallow water. *Computers & graphics 26*, 2, 291–307.

Glad, A. 2017. Real-time VFX Production Tips. *80.lv*. https://80.lv/articles/real-time-vfx-production-tips/.

Gonzato, J.-C. and Le Saëc, B. 2000. On modelling and rendering ocean scenes. *Journal of Visualization and Computer Animation 11*, 1, 27–37.

Jeschke, S., Skřivan, T., Müller-Fischer, M., Chentanez, N., Macklin, M., and Wojtan, C. 2018. Water Surface Wavelets. *ACM transactions on graphics 37*, 4, 94:1–94:13.

Jeschke, S. and Wojtan, C. 2017. Water wave packets. *ACM Transactions on Graphics 36*, 1–12. http://dx.doi.org/10.1145/3072959.3073678.

Kallin, D. 2008. Real time large scale fluids for games. *SIGRAD 2008. The Annual SIGRAD Conference Special Theme: Interaction; November 27-28; 2008 Stockholm; Sweden*,

Linköping University Electronic Press, 31–38.

Kass, M. and Miller, G. 1990. Rapid, stable fluid dynamics for computer graphics. *Proceedings of the 17th annual conference on Computer graphics and interactive techniques - SIGGRAPH '90*. http://dx.doi.org/10.1145/97879.97884.

Kellomäki, T. 2017. Fast Water Simulation Methods for Games. *Computers in Entertainment 16*, 1–14. http://dx.doi.org/10.1145/2700533.

Loviscach, J. 2002. A Convolution-Based Algorithm for Animated Water Waves. *Eurographics (Short Papers)*.

Mastin, G.A., Watterberg, P.A., and Mareda, J.F. 1987. Fourier Synthesis of Ocean Scenes. *IEEE computer graphics and applications 7*, 3, 16–23.

Max, N., Becker, B., and Crawfis, R. 1993. Flow volumes for interactive vector field visualization. *Proceedings Visualization '93*, 19–24.

O'Brien, J.F. and Hodgins, J.K. 1995. Dynamic simulation of splashing fluids. *Proceedings Computer Animation'95*, 198–205.

Ochoa, C.G. and Holder, D. 2012. Water technology of Uncharted. *Presentation in Game Developers' Conference*.

Ojeda, J. and Susín, A. 2013. Enhanced Lattice Boltzmann Shallow Waters for Real-time Fluid Simulations. *Eurographics (Short Papers)*, 25–28.

Ottosson, B. 2011. Real-time interactive water waves. https://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2011/rapporter11/ottosson_bjorn_11105.pdf.

Peachey, D.R. 1986. Modeling waves and surf. *Proceedings of the 13th annual conference on Computer graphics and interactive techniques - SIGGRAPH '86*. http://dx.doi.org/10.1145/15922.15893.

Schachter, B. 1980. Long crested wave models. *Computer Graphics and Image Processing 12*, 2, 187–201.

Selle, A., Fedkiw, R., Kim, B.M., Liu, Y., and Rossignac, J. 2007. An Unconditionally Stable MacCormack Method. *Journal of Scientific Computing 35*, 2-3, 350–371.

Tessendorf, J. 2004. Simulating ocean surfaces. *Part of ACM SIGGRAPH*.

Tessendorf, J. 2008. Simulation of Interactive Surface Waves. https://people.cs.clemson.edu/~jtessen/reports/papers_files/SimInterSurfWaves.pdf.

Tessendorf, J. 2014. eWave: Using an Exponential Solver on the iWave Problem. *Technical*

Note.

Thurey, N., Muller-Fischer, M., Schirm, S., and Gross, M. 2007. Real-time Breaking Waves for Shallow Water Simulations. *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, 39–46.

Torres, G. and Mike, S. 2012. Assassin's Creed III: The tech behind (or beneath) the action. *fxguide*. https://www.fxguide.com/fxfeatured/assassins-creed-iii-the-tech-behind-or-beneath-the-action/.

Ts'o, P.Y. and Barsky, B.A. 1987. Modeling and Rendering Waves: Wave-tracing Using Beta-splines and Reflective and Refractive Texture Mapping. *ACM transactions on graphics 6*, 3, 191–214.

Wang, H., Miller, G., Turk, G., and Turk, G. 2007. Solving General Shallow Wave Equations on Surfaces. *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 229–238.

Yu, J., Wojtan, C., Turk, G., and Yap, C. 2012. Explicit Mesh Surfaces for Particle Based Fluids. *Computer graphics forum: journal of the European Association for Computer Graphics 31*, 2pt4, 815–824.

Yuksel, C., House, D.H., and Keyser, J. 2007a. Implementing wave particles for real-time water waves with object interaction. *SIGGRAPH Sketches*, 14.

Yuksel, C., House, D.H., and Keyser, J. 2007b. Implementing wave particles for real-time water waves with object interaction. *ACM SIGGRAPH 2007 sketches on - SIGGRAPH '07*. http://dx.doi.org/10.1145/1278780.1278797.