# BREAKING WAVES IN SHORELINE WATER SIMULATIONS FOR GAMES

**Calum Leask**

**Abertay University**

**School of Arts, Media and Computer Games**

**May 2016**

# Abstract

3D water simulations in video games are becoming more and more realistic. One of the biggest challenges with simulating realistic water is the amount of processing power that is required, something games simply cannot afford to expend. Breaking waves are all too often neglected in water simulations as they are computationally expensive and usually have no impact on the gameplay. However, there is potential for shoreline water simulations to attain higher levels of realism by incorporating breaking waves. This project investigates and builds upon current techniques used for simulating breaking waves in order to determine how realistic a real-time simulation of shoreline water could be.

An application was developed, demonstrating a real-time simulation of shoreline water which incorporates breaking waves. The simulation uses the one dimensional shallow-water equations in order to simulate the flow of water towards the shoreline. Waves were located and tracked within the simulation so that meshes could be constructed to create the impression of overturning waves. Particles were generated from the meshes along the length of the wave to visually reflect the breaking waves. The simulation was evaluated with regards to performance and its visual appearance. Overall the simulation was found to be realistic, concluding that shoreline water simulations which incorporate breaking waves have the potential to feature in video games.

# Acknowledgements

I would like to take the opportunity to thank Dr Karen Meyer for her guidance and supervision during the course of the project. I would also like to thank all of the lecturers at Abertay University who have taught and inspired me throughout my undergraduate education. Finally I would like to thank all of my friends and family for their continual support and encouragement over the years.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# INTRODUCTION

Water covers 71% of our planet (NOAA 2016) and is present in our everyday lives therefore it should come as no surprise that water features in so many video games. However, simulating realistic water at real-time is a challenge for developers, even with today's processing power. Accurate water simulations are computationally expensive and water simulations in video games need to be fast and also look convincing.

Water is complex, it behaves and responds very differently to rigid bodies. It is made up of particles which are free to move independently of each other, describing nonlinear motion. Although the volume of water in a simulation may remain constant its topology can be constantly changing based on a whole host of external factors. Coupling real-time rigid body physics interactions with fluid simulations can also prove to be a real challenge in video games.

Accurate water simulations have existed in engineering for years using models such as the Navier-Stokes equations which are the basic equations that govern fluid dynamics (CFD Online 2012). However these equations, which strive for accuracy, are far too complex to solve for large 3D water simulations in video games at real-time. The equations need to be simplified or other techniques for simulating water need to be used. Grid based simulations are often used for large body water simulations like rivers, oceans and lakes whereas smaller features like

fountains, dripping taps and broken fire hydrants might use particle based simulations, animations or other more suited techniques.

Georges Torres, Senior Technical Director for Assassin's Creed III (2012), said "In feature films just a decade ago, rendering oceans was reason enough to seek an Oscar." (Seymour 2012). Comparing the jagged water's edge from Morrowind (2002) to the stunning oceans of Assassin's Creed Black Flag (2013) there is no doubt in the past decade water simulations have come a long way in video games (see Figures 1 and 2).

**Figure 1: Water from 'The Elder Scrolls III: Morrowind'**

**Figure 2: Ocean from 'Assassin's Creed IV: Black Flag'**

Efficiency is key in the simulation of water in video games. All the processing power available cannot be given solely to the water simulation as other areas of the game usually require a lot of processing power. This means the simulation should "be at least around 3–20 times faster" (Kellomäki 2012, p.11) than real-time, depending on the complexity of the game.

The believability and the photorealism of water in video games is also an important aspect of water simulations since games are becoming more and more photorealistic and water needs to keep up with the rest of the graphics to avoid looking out of place. One area where water lacks realism in video games is the absence of breaking waves as waves approach the shoreline. "Real breaking waves at the shore that curl over themselves is a very tough problem and generally isn't implemented." (Kayne 2014). Games can and do get away with ignoring breaking waves since it is down to environmental conditions that govern the size of waves and whether they will break and come crashing towards shore. No one is going to complain if a game doesn't include breaking waves, but it would look nice and it would definitely take shoreline water simulations to the next level in video games.


**Figure 3: Shoreline water in 'The Legend of Zelda: The Wind Waker'**

The Legend of Zelda: The Wind Waker (2002) is an early example of a game which included a subtle interaction between the sea and the shore with advancing and receding foam (see Figure 3). Super Mario Sunshine (2002) also included shoreline interaction along with numerous other visual effects and was hailed for having beautiful water for its time.

As hardware has evolved over the last decade shoreline water simulations have also improved as shown by the dramatic difference between the beach water in Grand Theft Auto (GTA): San Andreas (2004) for PlayStation 2 and the water in GTA V (2014) for PlayStation 4 (see Figure 4).

**Figure 4: Shoreline water in 'GTA: San Andreas' (PS2) vs 'GTA V' (PS4)**

Adding breaking waves to real-time water simulations has the potential to greatly improve the realism of water within video games and increase the wow factor associated with beautiful water simulations, but this is so often neglected due to the complexity breaking waves add to the water simulations (Rath 2014). This project will seek to investigate the area of breaking waves along with shoreline water simulations in video games by attempting to answer the following question:

## Research Question

*"How realistic can a simulation of shoreline water be in a real-time application which also incorporates breaking waves?"*

## Aims

The aims of the project are to:

- Determine the best approach to incorporating breaking waves into an efficient and realistic simulation of shoreline water.
- Produce a realistic simulation of shallow water that interacts with a shoreline in a real-time application.

## Objectives

- Research how shallow tidal water interacts with shorelines and under what conditions waves break.
- Investigate existing and possible techniques for simulating shallow water and breaking waves and how they can be combined.
- Create an application which simulates shallow water that interacts with a shoreline and includes the capacity for breaking waves using the researched techniques.
- Determine how realistic the simulation is whilst running at real-time and investigate areas where the simulation could be improved.

# Chapter 1. Context

## 1.1 Background

As hardware has advanced over the last two decades so have the 3D water simulations that are found in video games. The early nineties saw the rise in 3D video game development which paved the way for the development of 3D water implementations. 3D water started out as statically textured or coloured flat planes, as witnessed in Doom (1993). Implementations slowly progressed as motion was simulated by using moving texture coordinates as seen in games such as Shadow Warrior (1997) and GoldenEye 007 (1997). Visually, water became more appealing when transparency was incorporated through alpha blending, which allowed players to see below the surface of the water and through other features like waterfalls. Surface details such as splashes and ripples began to emerge in games like Super Mario 64 (1996), conveying player's interactions with the water. A huge step was taken towards simulating waves when the game Wave Race 64 (1996) replaced traditional planar water with a height field, which was able to replicate waves passing through water.

Even with all the advancements made in the past two decades realistic water simulation is still a substantial challenge in the development of video games (Kotzer 2015). Extensive research has been done over the years and is still ongoing, looking into different methods and techniques for simulating and rendering a variety of different types of water within in a gaming context. There are now many different methods available for simulating water in video games which Kellomäki (2012) provides a comparison of. The nature and requirements of the water to be simulated determines the appropriate techniques to be used.

## 1.2 The One Dimensional Shallow-Water Equations

To model and simulate shoreline water we can consider the water to be an ideal fluid which has no viscosity and the density of the fluid remains constant. This allows us to work with the Euler equations (see Appendix A) which are a simplification of the Navier-Stokes equations. A further simplification can be applied if we ignore the vertical component of acceleration which gives us the shallow-water equations (Randall 2006):

$$\frac{\delta \boldsymbol{u}}{\delta t} + \boldsymbol{u}\frac{\delta \boldsymbol{u}}{\delta x} = -g\frac{\delta \boldsymbol{h}}{\delta x} \qquad (1)$$

$$\frac{\delta \boldsymbol{h}}{\delta t} + \boldsymbol{u}\frac{\delta \boldsymbol{h}}{\delta x} + \boldsymbol{h}\frac{\delta \boldsymbol{u}}{\delta x} = 0 \qquad (2)$$

Where $\boldsymbol{u}$ and $\boldsymbol{h}$ are functions of $x$ and $t$ alone. The first equation is derived from Newton's 2nd Law $F = ma$. The second equation, known as the continuity equation, is derived from the incompressibility condition and states that a particle on the surface of the fluid does not separate from the rest of the fluid.

These simplifications greatly reduce the complexity and therefore the computational expense of the simulation. This makes the shallow-water equations more suitable for simulating water in real-time applications such as games. Since the simulation is only concerned with waves travelling in a single direction towards the shore, the one dimensional shall-water equations are sufficient to model the basics of shoreline water. They are also far less computationally expensive to compute than the more complex two dimensional shallow-water equations.

However, there are limitations of using the shallow-water equations to model shoreline water. Since the equations do not account for three-dimensional flows and because of the continuity equation, breaking waves cannot be modelled by the

shallow-water equations alone. Although the shallow-water equations do account for interactions between the water and boundaries and dry zones such as a shoreline as the water level approaches zero.

## 1.3 Eulerian and Lagrangian Viewpoints

Two common approaches exist to discretize the shallow-water equations and track the motion of the shallow water. They are the Eulerian (grid based) and the Lagrangian (particle based) approaches (Bridson and Müller-Fischer 2007, p. 4).

The Eulerian viewpoint uses a grid to track the flow of the water, calculating and storing the height and velocity of the water at fixed cells in the grid as the water flows through the grid. Eulerian schemes are easy to work with and retain the uniform regularity of the grid. However for an Eulerian scheme to maintain stability small time steps are required which impacts performance time.

The Lagrangian viewpoint treats the water as a system of particles, calculating each particle's position, height and velocity as they move and interact with surrounding particles. Lagrangian schemes allow for larger time steps as they are less restricted with stability requirements. However, the biggest drawback of Lagrangian schemes is they can become computationally expensive to calculate for a larger number of particle-particle interactions as the number of particles in a system increases. Also, particles which started regularly spaced usually become irregularly spaced as they move over time which can lead to a loss in global accuracy.

## 1.4 Semi-Lagrangian Schemes

Semi-Lagrangian schemes have been developed to combine the advantages of both Eulerian and Lagrangian schemes and were first introduced by Robert (1981). Semi-Lagrangian schemes use an Eulerian framework of a regular grid whilst

taking the discrete equations from the Lagrangian viewpoint. At every time step the origin of particles at fixed cells in the grid are calculated by integrating back along particle trajectories. The height and velocity of the particles at their origin can then be estimated by interpolating the values from the surrounding grid points. Both Bates et al. (1993) and Wong et al. (2013) demonstrate how semi-Lagrangian schemes can be utilised to model the shallow-water equations. Layton and Van de Panne (2002) implemented a graphical representation (see Figure 5) to demonstrate their 'numerically efficient and stable algorithm for animating water waves' which takes a semi-Lagrangian approach.

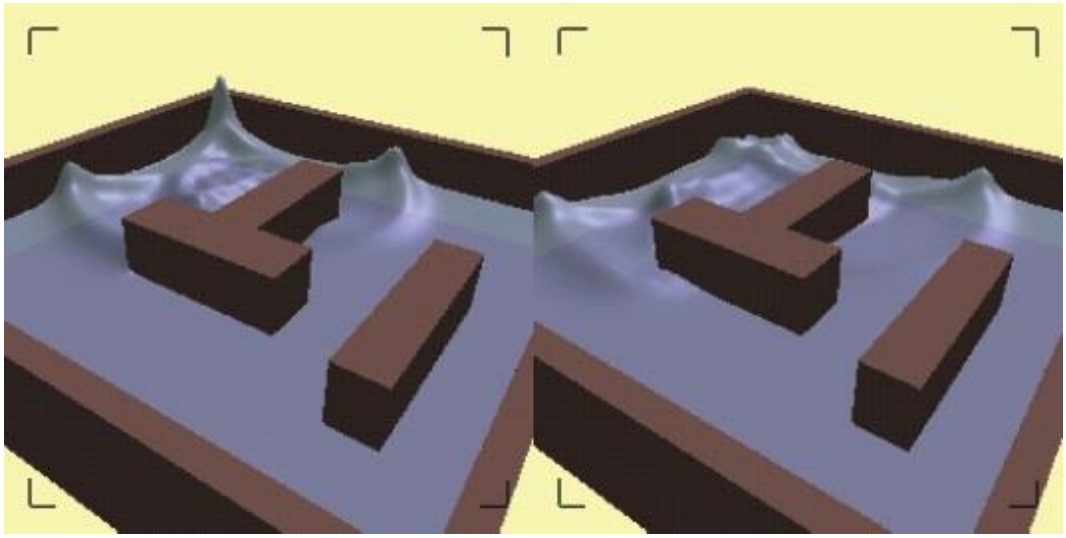

**Figure 5: Layton and Van de Panne's (2002) animated water waves**

## 1.5 Related Work

Implementing water that includes waves with the capability of breaking and crashing poses a difficult challenge for video game developers for a few reasons. Eulerian Grid based simulations that use height-fields with a mesh to represent water suffer from the problem that a mesh which is manipulated by a height-field

cannot fold over itself to allow a wave to break. On the other hand Lagrangian particle based simulations are too resource intensive and expensive to compute for large bodies of water in video games.

Research has already been carried out looking at real-time breaking waves in water simulations. Gross et al. (2007) incorporate breaking waves into their water simulation, which uses the shallow-water equations, by generating wave patches which are a mesh of connected particles projected from the peak of the waves. Bruan, Raupp Musse and Strube de Lima (2010) take a different approach, generating breaking waves by manipulating and animating the vertices of waves as they travel through a 2D mesh. Both examples use the more computationally efficient Eulerian grid based approaches as a starting point for modelling the water, then build upon the simulation to include breaking waves. Particles are also used in both examples to emulate the visual effects of foam and splashing from the waves as they begin to break and crash.

## 1.6 Photorealistic Water

Photorealistic water is not as easy to render as simply applying a texture to an object. Water reflects, it refracts, it ripples, it sprays, it splashes and it foams. Most of these details are important, depending on the nature of the water, and should be considered to effectively render realistic water simulations which are convincing and visually appealing (Kayne 2014). A more in depth discussion and analysis of the different water simulation techniques and methods for rendering are given in the paper 'A Survey of Ocean Simulation and Rendering Techniques in Computer Graphics' (Crespin et al. 2011).

In the context of the project, where shoreline water that incorporates breaking waves is the focus, the following visual aspects will need to be implemented to maximise the photorealism of the simulation:

- Lighting
- Reflection
- Ripples
- Splashes
- Foam

Refraction and water caustics are not going to be considered since the focus is on the water's surface which will be turbulent, preventing viewers from viewing below the surface of the water in any detail.

# Chapter 2. Methodology

Having gained an understanding of the complexity behind realistic simulations of shallow water this chapter will now discuss the methods taken to produce a realistic simulation within a real-time an application. The application has been developed with Visual Studio 2013 in C++ and using DirectX11. The application allows the user to adjust the heights of waves traveling towards a shore and watch them break at the shoreline.

## 2.1 Representing the Water's Surface

The surface of the water is represented by a 2D height field of a size 250x200 in the form of a 2D array of vertices. Each vertex contains three-component position and normal data. The normal data is used to correctly light the water and to calculate reflections on the water's surface. The heights of the vertices in the height field will be manipulated by the one-dimensional shallow-water equations, with water flowing in the direction perpendicular to the shoreline (see Figure 6).

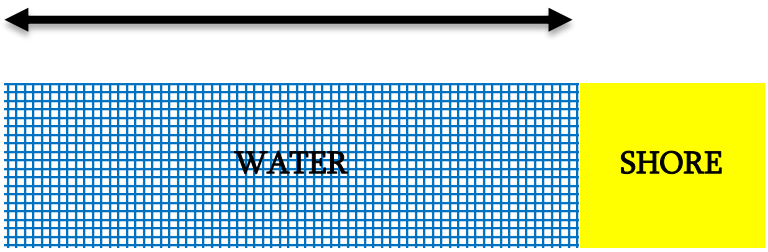


**Figure 6: Direction of the flow of water on the 2D height field**

## 2.2 Numerically Solving the 1D Shallow-Water Equations

The approach taken to spatially discretise the shallow-water equations was an implicit semi-Lagrangian time integration method, since it is shown to be highly efficient for real-time applications as it remains stable for large time steps whilst also remaining capable of producing realistic waves. Adding a ground height to the shallow-water equations gives us equations (1) and (2) where $b$ is the height of the ground and $\boldsymbol{h}$ is still the height of the height field. It is important to take into account the height of the ground as the simulation deals with a sloping shoreline.

$$\frac{\delta \boldsymbol{u}}{\delta t} + \boldsymbol{u}\frac{\delta \boldsymbol{u}}{\delta x} = -g\frac{\delta \boldsymbol{h}}{\delta x} \qquad (1)$$

$$\frac{\delta \boldsymbol{h}}{\delta t} + \boldsymbol{u}\frac{\delta(\boldsymbol{h} - b)}{\delta x} + (\boldsymbol{h} - b)\frac{\delta \boldsymbol{u}}{\delta x} = 0 \qquad (2)$$

Rewritten in Lagrangian form and using $\boldsymbol{d}$ as the depth of the water, $\boldsymbol{d} = \boldsymbol{h} - b$, the shallow-water equations become:

$$\frac{d\boldsymbol{u}}{dt} + g\frac{\delta \boldsymbol{h}}{\delta x} = 0 \qquad (3)$$

$$\frac{d\boldsymbol{h}}{dt} - \boldsymbol{u}\frac{\delta b}{\delta x} + \boldsymbol{d}\frac{\delta \boldsymbol{u}}{\delta x} = 0 \qquad (4)$$

In a semi-Lagrangian scheme we calculate the derivatives from the trajectories of particles at positions $x_i$ at a time of $t^{n+1}$ and its position at a time of $t^n$ which is the departure point of the particle, denoted as $\tilde{x}_i^n$.

$$\tilde{x}_i^n = x_i - \Delta t \boldsymbol{u}^n(x_i) \qquad (5)$$

With this we can approximate the Lagrangian derivatives with equations (6) and (7) where $\widetilde{\boldsymbol{u}}$ and $\widetilde{\boldsymbol{h}}$ are the velocity and height of the particle at departure point $\tilde{x}_i^n$.

$$\frac{d\boldsymbol{u}}{dt} = \frac{\boldsymbol{u}^{n+1} - \widetilde{\boldsymbol{u}}^n}{\Delta t} \qquad (6)$$

$$\frac{d\boldsymbol{h}}{dt} = \frac{\boldsymbol{h}^{n+1} - \widetilde{\boldsymbol{h}}^n}{\Delta t} \qquad (7)$$

We use quadratic interpolation to calculate $\widetilde{\boldsymbol{u}}$ and $\widetilde{\boldsymbol{h}}$ at departure point $\tilde{x}_i^n$ by considering neighbouring heights and velocities, since $\tilde{x}_i^n$ may not lie exactly on a grid point. By substituting equations (6) and (7) into equations (3) and (4) we obtain equations (8) and (9).

$$\frac{\boldsymbol{u}^{n+1} - \widetilde{\boldsymbol{u}}^n}{\Delta t} + g\frac{\delta \boldsymbol{h}^{n+1}}{\delta x} = 0 \qquad (8)$$

$$\frac{\boldsymbol{h}^{n+1} - \widetilde{\boldsymbol{h}}^n}{\Delta t} - \boldsymbol{u}^{n+1}\frac{\delta b}{\delta x} + \boldsymbol{d}^n\frac{\delta \boldsymbol{u}^{n+1}}{\delta x} = 0 \qquad (9)$$

We can solve equation (9) by eliminating the terms $\boldsymbol{u}^{n+1}$ and $\frac{\delta \boldsymbol{u}^{n+1}}{\delta x}$. Rearranging (8) we can get $\boldsymbol{u}^{n+1}$ as shown in equation (10). By taking the derivative of equation (10) with respect to $x$ we can get $\frac{\delta \boldsymbol{u}^{n+1}}{\delta x}$ as shown in equation (11).

$$\boldsymbol{u}^{n+1} = \widetilde{\boldsymbol{u}}^n - \Delta t g\frac{\delta \boldsymbol{h}^{n+1}}{\delta x} \qquad (10)$$

$$\frac{\delta \boldsymbol{u}^{n+1}}{\delta x} = \frac{\delta \widetilde{\boldsymbol{u}}^n}{\delta x} - \Delta t g\frac{\delta^2 \boldsymbol{h}^{n+1}}{\delta x^2} \qquad (11)$$

By substituting equations (10) and (11) into equation (9) we arrive at the following differential equation without any unknown velocity terms.

$$\boldsymbol{h}^{n+1} + \Delta t^2 g \frac{\delta b}{\delta x} \frac{\delta \boldsymbol{h}^{n+1}}{\delta x} - \Delta t^2 g \boldsymbol{d}^n \frac{\delta^2 \boldsymbol{h}^{n+1}}{\delta x^2} = \widetilde{\boldsymbol{h}}^n + \Delta t \widetilde{\boldsymbol{u}}^n \frac{\delta b}{\delta x} - \Delta t \, \boldsymbol{d}^n \frac{\delta \widetilde{\boldsymbol{u}}^n}{\delta x} \qquad (12)$$

Using the central difference formula we can spatially discretised equation (12) so that the one-dimensional shallow-water equations are reduced to the following differential equation which we can then solve:

$$\boldsymbol{h}_i^{n+1} + \Delta t^2 g \left(\frac{b_{i+1} - b_{i-1}}{2\Delta x}\right)\left(\frac{\boldsymbol{h}_{i+1}^{n+1} - \boldsymbol{h}_{i-1}^{n+1}}{2\Delta x}\right) - \Delta t^2 g d_i^n \left(\frac{\boldsymbol{h}_{i-1}^{n+1} - 2\boldsymbol{h}_i^{n+1} - \boldsymbol{h}_{i+1}^{n+1}}{\Delta x^2}\right)$$
$$= \widetilde{\boldsymbol{h}}_i^n + \Delta t \widetilde{\boldsymbol{u}}_i^n \left(\frac{b_{i+1} - b_{i-1}}{2\Delta x}\right) - \Delta t \boldsymbol{d}_i^n \left(\frac{\widetilde{\boldsymbol{u}}_{i+1} - \widetilde{\boldsymbol{u}}_{i-1}}{2\Delta x}\right) \qquad (13)$$

Where $\boldsymbol{h}^{n+1}$ are the unknown heights of the water along the one-dimensional line of water which we are trying to calculate at a time of $t^{n+1}$. $g$ denotes the constant gravitational force acting upon the simulation and $b$ denotes the height of the ground, which remains constant throughout the simulation for every point $i$. $\boldsymbol{d}^n$ is the depth of the water at a time of $t^n$, which is a simplification for $\boldsymbol{d}^n = \boldsymbol{h}^n - b$.

Equation (13) can then be solved using the conjugate gradient method (Hestenes and Stiefel 1952) as it is effective for solving large and sparse systems efficiently. For every time step, the heights $\boldsymbol{h}^{n+1}$ at time $t^{n+1}$ are calculated. Using these heights the velocities $\boldsymbol{u}^{n+1}$ at time $t^{n+1}$ can then be calculated from equation (10).

To keep equation (13) stable the following time step $\Delta t = 1/60s$ and step size $\Delta x = 0.1m$ were used.

## 2.3 The Sea Bed

As mentioned earlier, the simulation incorporates a sloped sea bed (see Figure 7). This is important as the presence of a sloping sea bed naturally causes waves to increase in height as they approach the shore, where the depth of the water approaches zero (Surfing Waves 2011?).

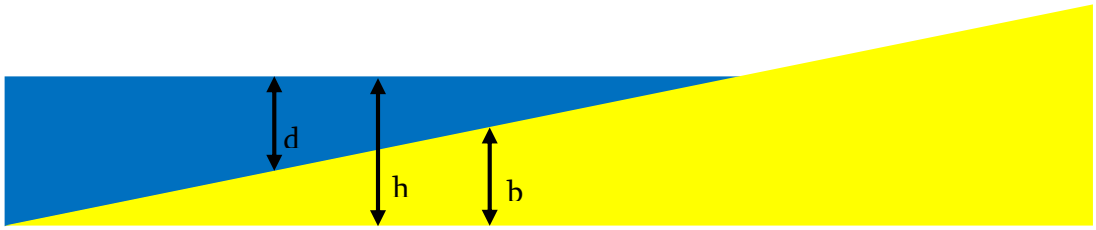


**Figure 7: A gradually increasing sea bed**

## 2.4 Boundary Conditions

We have two boundary conditions to consider within the simulation. The firs is an open boundary out in the ocean at position $x_0$, where waves will enter into the simulation. The second is a free moving boundary which represents the shoreline at position $x_s$ (see Figure 8). Since the water is free to flow over the sea bed there should be a naturally occurring movement of this boundary as the shoreline advances and recedes due to the influence of incoming waves. The height and velocity values must be carefully considered at these two boundaries so that the simulation remains stable.

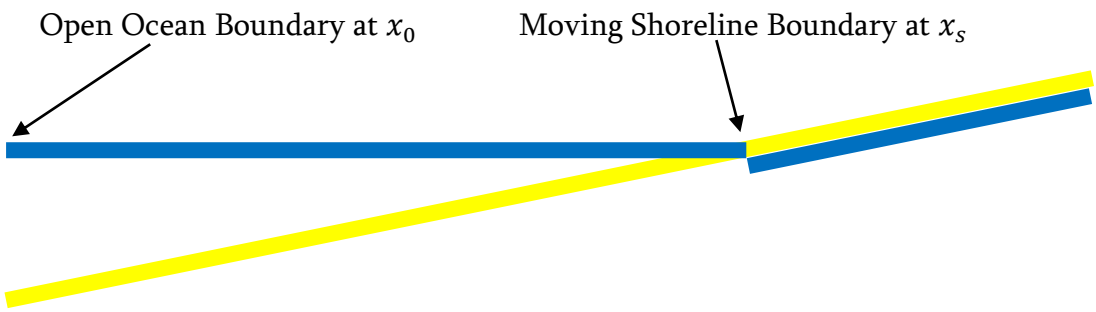


**Figure 8: Side view of the simulation with the two boundaries**

### 2.4.1 Open Ocean Boundary

When calculating numerical derivatives using the central difference formula at positions $x_i$, values are required at $x_{i-1}$ and $x_{i+1}$. When considering the boundary out in the ocean at position $x_0$ we realise no velocity or height values exists at position $x_{-1}$ so we must estimate these values ourselves. Since water is generally flowing into the simulation at this boundary it is safe to assume the particle outside the boundary at position $x_{-1}$ has the same departure velocity as the particle on the boundary at position $x_0$ so that $\widetilde{\boldsymbol{u}}_{-1} = \widetilde{\boldsymbol{u}}_0$. It is also safe to assume that the height of the water outside the boundary at position $x_{-1}$ and time $t^{n+1}$ will be similar to the height of the water on the boundary at position $x_0$ and time $t^n$ so we can make them equal $\boldsymbol{h}_{-1}^{n+1} = \boldsymbol{h}_0^n$. We can also say that the ground level at position $x_{-1}$ is equal to the ground level at position $x_0$, $b_{-1} = b_0$, since the ground level is constant throughout the simulation.

### 2.4.2 Moving Shoreline Boundary

The second boundary in the simulation is the shoreline at position $x_s$ and is not a fixed boundary. This boundary is not treated the same as the boundary at $x_0$ since velocity and height values beyond the boundary do exist. The 2D height field does not just cover the region from the boundary at $x_0$ to the shoreline at $x_s$, it stretches

beyond $x_s$ and up the shore to allow for the movement of the shoreline. The depth of the water at positions greater than $x_s$ is very small and therefore negligible since no water is present at these positions. To reduce numerical error, the velocity at positions greater than $x_s$ is forcibly set to zero. All positions on the height field greater than $x_s$ are naturally hidden beneath the shore (see Figure 8) since the depth of the water is very small and the mesh of the sea bed is raised by a small value to account for this.

## 2.5 Interactions with the Shoreline

For simplicity the whole range of the height field is not considered whilst solving the shallow-water equations to obtain updated height and velocity values since the region above the shoreline contains no water. Only the region from $x_0$ to $x_s$ is considered. At a time $t^n$ the water depth at position $x_{s-1}$ will be greater than zero and at position $x_s$ the water depth will be zero (see Figure 9). The height and velocity at position $x_{s+1}$ will both be zero as there is no water present to the right hand side of the shoreline. Depending on the rest of the simulation, at time $t^{n+1}$ the height of the water at position $x_s$ may rise to become greater than zero so that the shoreline boundary advances from position $x_s$ to $x_{s+1}$. The same also applies for a receding shoreline, where the height of the water at time $t^{n+1}$ and position $x_{s-1}$ may drop to zero so that the shoreline boundary recedes from position $x_s$ to $x_{s-1}$.

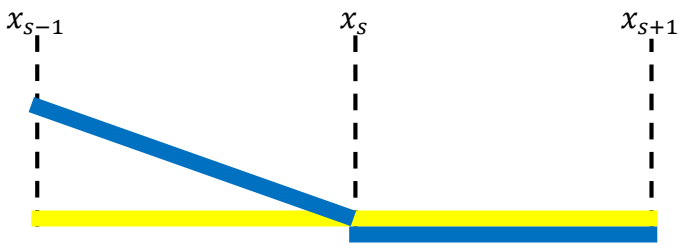


**Figure 9: Shoreline boundary at $x_s$**

Although the depth of the water at position $x_s$ is zero this does not mean the velocity at this position should be zero. From Figure 9 it can be seen that water just touches position $x_s$. At the next time step a particle at position $x_s$ will be considered and its departure point $\tilde{x}_s^n$ will need to be calculated. This requires the particle to have a velocity unless the water is at complete rest, in which case the shoreline will not move.

The shoreline is a special case boundary where continual wetting and drying occurs either side of the boundary as the water advances and recedes over the sea bed. This transition should be smooth so that it is realistic. However the movement of the shoreline can be prone to sticking and is often jittery. To solve this, the velocity of the water at the boundary $x_s$ is adjusted, using values from Khan and Lai (2014), to keep the movement of the shoreline smooth. When $\boldsymbol{h}_{s-1}^n > \boldsymbol{h}_s^n$, the height of the water at $x_{s-1}$ is greater than the height of the water at $x_s$ then the following amount is added to the velocity at $x_s$ to keep the shoreline moving smoothly:

$$\boldsymbol{u}_s^n \mathrel{+}= 2\sqrt{g(\boldsymbol{h}_{s-1}^n - \boldsymbol{h}_s^n)}$$

## 2.6 Propagating Waves

In order to create breaking waves in a shoreline water simulation, waves must first be propagated into the simulation. This is done by periodically applying harmonic motion to adjust the height of the water at the open boundary at $x_0$, which sends sine waves into the simulation (see Figure 10). These waves gradually steepen as they approach the shore. The height of the waves propagated into the simulation

can be adjusted along with the frequency of their propagation. Different frequencies and different heights of waves have different effects on the simulation as a whole and determine if/when a wave will become steep enough to break.

**Figure 10: Wave propagation from the open boundary**

## 2.7 Locating and Tracking Waves with Potential to Break

Whether waves within the simulation will break or not is determined by the steepness of their front. All of the waves in the simulation are located by sweeping across the height field from position $x_s$ to $x_0$, finding positions with sufficiently steep gradients which then become the fronts of waves that have the potential to break (see Figure 11).

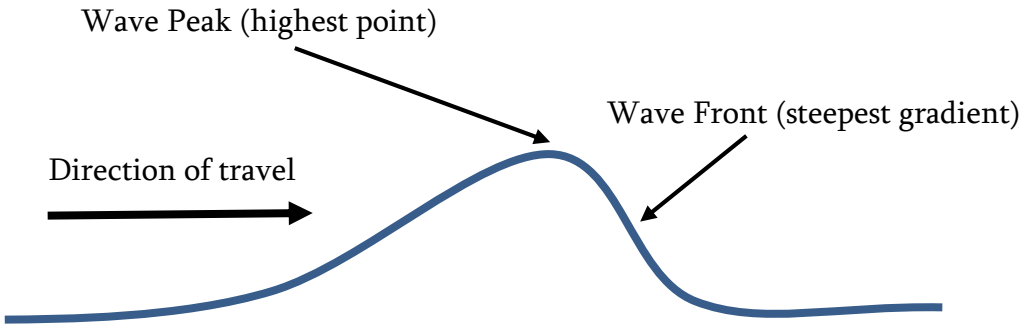


**Figure 11: A wave with the potential to break**

The peak position of each wave is then located by finding the highest point behind the wave front. Since all waves travel in the same direction, towards the shore, the peak of the wave will always be behind. Since we know the direction a wave is travelling, we can use advection to transport this location forward to more quickly determine the updated position of the wave front for every time step of the simulation. The updated wave front position is then used to update the position of the peak of the wave.

## 2.8 Creating and Animating Breaking Waves

Since the shallow-water equations do not account for three-dimensional flows they cannot model features such as breaking waves which overturn. We must account for this and add the functionality of breaking waves on top of the simulation. Having previously detected waves with the potential to break we can periodically project particles forward from the peak of the wave to form a mesh of water to represent the overturning wave (see Figures 12 and 13).

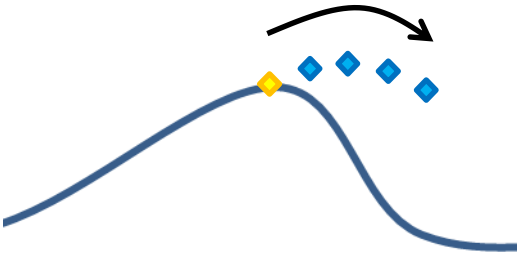

**Figure 12: Particles spawn from the peak of waves**

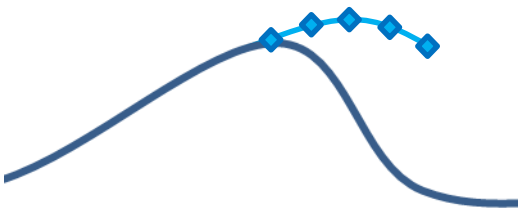

**Figure 13: Mesh formed by connecting projected particles**

Particles are projected from the peak of the wave with a forward velocity directly proportional to the steepness of the wave front. Particles are given an initial vertical velocity directly proportional to the difference in height between the

water at position $x_p$ and $x_{p-1}$, where $x_p$ is the position at the peak of the wave. This means that a wave whose peak is flatter will project particles with a smaller vertical velocity. The position of these particles is relative to the peaks from which they were spawned so that they move along with the wave. The velocities of particles are unaffected by the shallow-water equations and are only affected by the force of gravity.

As seen in Figure 13 a mesh of water is created along the wave to emulate the overturning nature of the wave. This mesh of water is created by connecting the projected particles, starting from the peak of the wave, and creating vertices for the mesh at these points. As the particles move through space the vertices of the mesh are appropriately updated so the mesh is constantly changing. If a wave slows down and its peak starts to recede then the wave has lost its potential to break so will not project any more particles. This leads to the current mesh of particles falling and crashing back into the height field.

## 2.9 Merging Breaking Waves into the Surface of the Water

Once the particle at the end of the mesh collides with the surface of the water (see Figure 14) there is the problem of deciding how the overturning wave should best flow back into the surface of the water. The mesh should blend seamlessly into the height field to ensure the breaking wave and the surface of the water appear as one. This has been achieved by raising all of the points on the height field which are under the mesh up to the height of the mesh and then removing the mesh (see Figure 15). The velocity of the water at these points is then set to the value of half of the velocity of the water at the peak of the wave. This keeps the water flowing smoothly towards the shore and also looks natural.

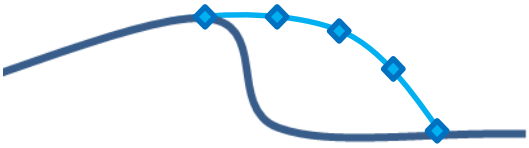

**Figure 14: The furthest particle in the mesh hitting the surface of the water**

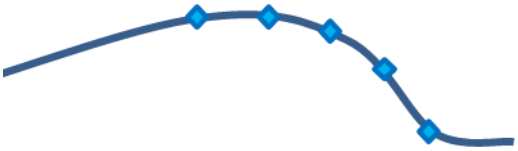

**Figure 15: Height field raised to join the particles in the mesh**

Raising the height field does introduce more water into the simulation, however it will not lead to an increase in the volume of water in the simulation over long periods of time. The height of the water is maintained at a constant level (1 metre) at the open boundary so any excess water will naturally leave the simulation. The temporary increase in water is not noticeable, especially since there is a constant flow of water which causes the shoreline to continually advance and recede.

## 2.10 Making the Simulation Photorealistic

As explained earlier, enhancing the photorealism of the simulation is one of the toughest challenges dues to the complexity of water and its behaviour, especially when dealing with breaking waves. In a simple and calm ocean water simulation, photorealism is not too difficult to achieve. However a shoreline water simulation with waves that approach the shore and break proves a tougher challenge when striving for photorealism. This is due to the added complexity that breaking waves bring to the simulation, where splashes and foaming of the water now need to be considered when looking to make the simulation photorealistic.

### 2.10.1 Colour and Surface Reflections

To produce photorealistic water it must reflect the surrounding environment, and it should also be coloured correctly. The colour given to the water comes from a blend of the reflected sky and a defined water colour. This blend is created from the Fresnel effect which states that the amount of reflection seen in a surface is dependent on the angle the surface is viewed at (Bim 2001). Reflecting the sky also emulates the effect of specular lighting where the sun is reflected, causing the water to glisten in the sun light.

To produce accurate reflections the three-component normal vectors to the surface at every vertex on the height field must be calculated. Calculating the normal vectors for every vertex in three-dimensional space at every frame can become very computationally expensive. However, due to the nature of the simulation, the normal vectors at each vertex will be constant along grid lines parallel to the shoreline as the heights are equal along these grid lines. This significantly reduces the number of calculations that would have been required if the simulation used the more complex two-dimensional shallow-water equations.

To add detail to the surface of the water a technique called 'Normal Mapping' has been used. This technique uses the red, green and blue values from a texture (see Figure 16) to manipulate the x, y and z components of normal vectors, faking the appearance of bumps on the surface of the water.

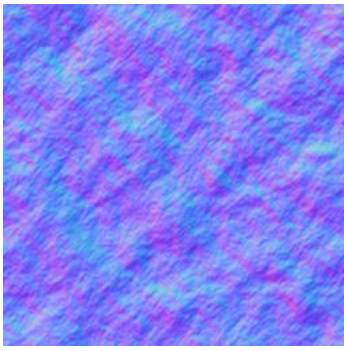

**Figure 16: Normal map which adds detail to the water's surface**

Figures 17 and 18 demonstrate the difference that normal mapping makes to the surface of the water in helping to make the water more realistic.

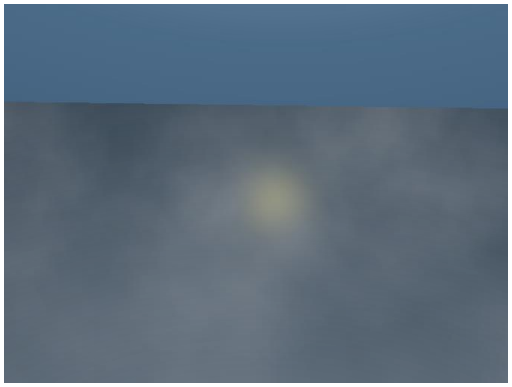

**Figure 17: Flat water surface**

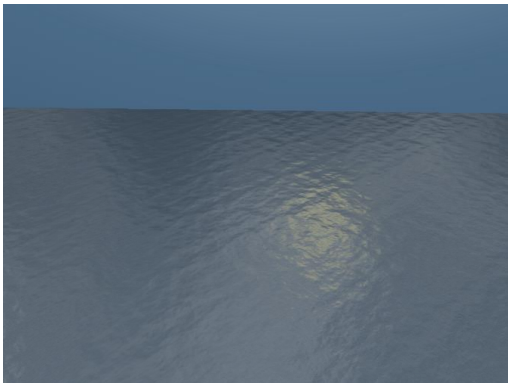

**Figure 18: Normal mapped water surface**

### 2.10.2 Animated Ripples

With surface detail added to the water through normal mapping the detail appears very static, which does not look convincing when waves pass through the height field. This could be improved by translating the normal map texture over time, but this would not be convincing. To solve the problem of a static normal map, animated ripples have been used to manipulate the texture coordinates of the normal map to give the appearance of unique ripples throughout the simulation (see Figure 19).

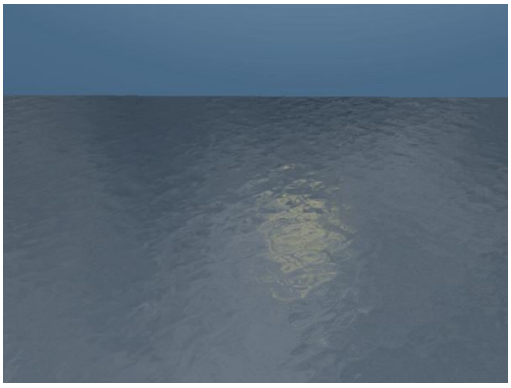

**Figure 19: Ripples animated throughout the water's surface**

To create the impression of unique animated ripples across the surface of the water a smaller repeating height field of size 40x40 has been created. Points within the height field are initially set to a height of zero. Every frame, points are then randomly chosen to be offset by a small value. By utilising and solving the two-dimensional dampened wave equation (see Appendix B) these offset points disperse through the height field creating ripples. Boundary values are not required for solving the dampened wave equation since the 40x40 height field loops round on itself.

Directly applying and repeating these ripples across the surface of the water would give the impression rain was falling onto the water. Instead of directly applying the ripples to the surface of the water, the normal vectors of the ripples are used to offset the texture coordinates of the normal map. This causes an animated disturbance within the normal map to successfully create a realistic ripple effect that might be witnessed at the shore. The ripples are also used to slightly modify the heights on the height field and the heights of the vertices which make up a breaking waves mesh. This is so that the shoreline and the edge of breaking waves do not appear completely straight, which would look unnatural.

### 2.10.3 Shoreline Foam

When a shoreline advances and recedes due to incoming waves, foam is produced along the shoreline. To create a foaming effect, each point in the height field along the line perpendicular to the shoreline stores a value based on the amount of foam present at that point. This value is kept between zero and one, where zero equates to there being no foam.

When a wave breaks and crashes back into the surface of the water, foam is applied to the points surrounding where the wave touched the surface by setting the values of the amount of foam to one. A layer of gradually fading foam is spread across the shoreline by using the code from Figure 20.

```
for (int position = 0; position < foam_width; ++position)
{
  if (foam[shoreline - position] < (foam_width - position) / foam_width)
  {
    foam[shoreline - position] = (foam_width - position) / foam_width;
  }
}
```

**Figure 20: Code which applies foam along the shoreline**

Foam is dispersed across the water with the following function:

$$f_i^{n+1} = \frac{5f_i^n + 3f_{i-1}^n + 2f_{i+1}^n}{10}$$

Where $f_i$ is the amount of foam at a point $i$ at a time step $n$. This function disperses the foam with a bias towards the left, meaning that foam will disperse more quickly out in the direction away from the shore and will linger for longer towards the shore. With an advancing shoreline, more foam will be produced since it takes longer to disperse from where the shoreline previously existed (see Figure 21). With a receding shoreline, foam is brought back with the shoreline (see Figure 22). This is all due to the bias in the above equation.

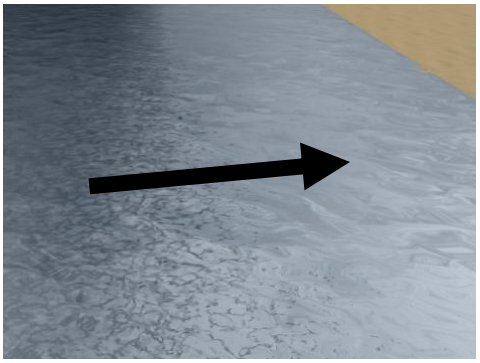

**Figure 21: Foam produced by an advancing shoreline**

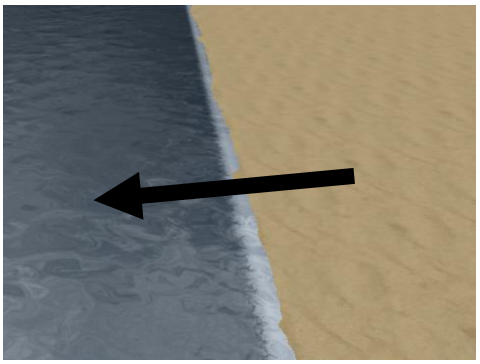

**Figure 22: Foam brought back by a receding shoreline**

A texture (see Figure 23) is used to add detail to the foam to aid in its realism. In the same way ripples were applied to the normal texture they are also applied to the foam texture to give the impression that the foam is moving and is affected by movement on the surface of the water.

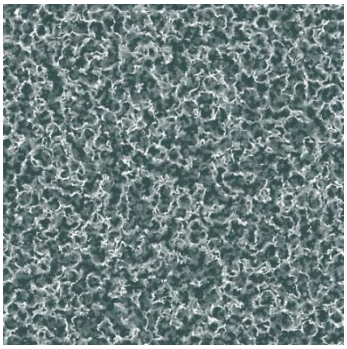

**Figure 23: Foam texture used to add detail to the foam**

The detail is not applied across all of the foam but is tethered towards the edges of the foam. By comparing Figures 24 and 25 it is shown that adding detail to the foam makes a substantial improvement in its appearance.

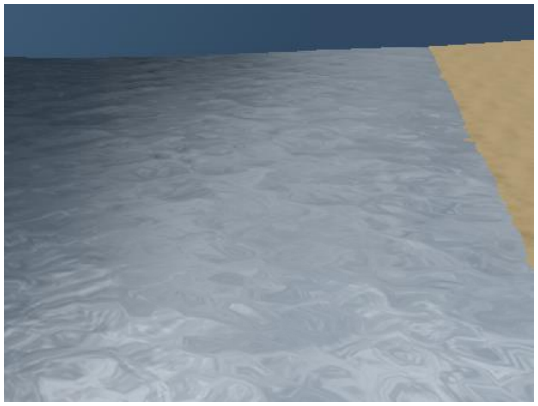

**Figure 24: Plain white foam**

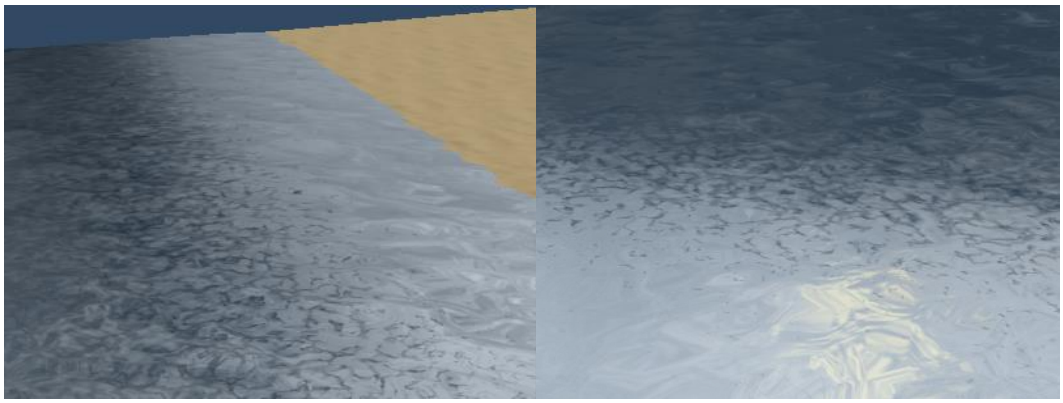

**Figure 25: Improved foam with added detail**

Generally when a wave breaks it also foams. To help a breaking wave stand out and to stop it from visually blending in with the rest of the water, foam is applied to the mesh which represents the overturning water (see Figure 26).

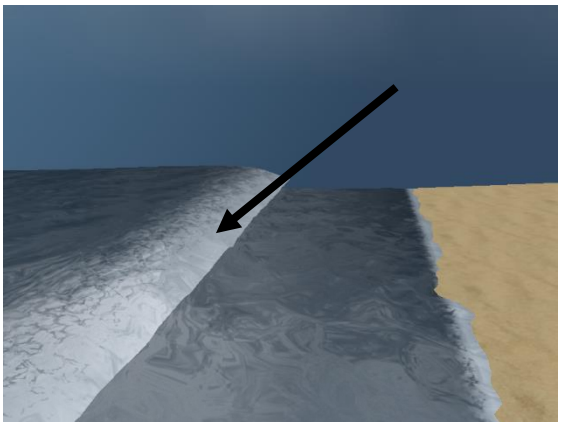

**Figure 26: Foam applied to the mesh of a breaking wave**

### 2.10.4 Splash Particle Effects

The final visual features of the water left to consider are splashes and spray. Semi-transparent particles have been used to emulate these visual effects. When a wave crashes back into the surface of the water, particles are spawned across the wave where the mesh of the wave touches the water (see Figure 27). These particles are given a forward velocity proportional to the velocity of the wave and an upward velocity proportional to the downward velocity of the particle which makes up the end of the wave mesh.

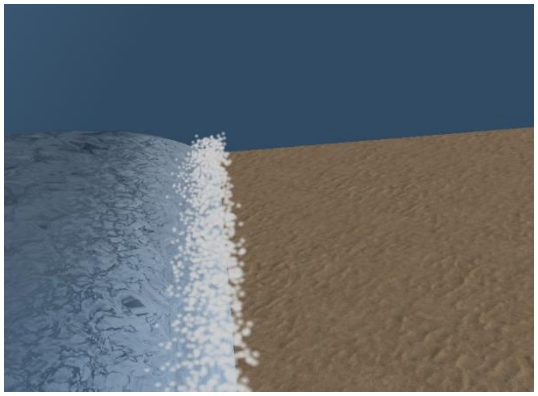

**Figure 27: Particles spawned where a wave crashed**

Throughout the process of a wave breaking, from start to end, particles are spawned along the wave mesh (see Figure 28). This livens up the breaking wave, helping to make it more realistic and better resemble a breaking wave.

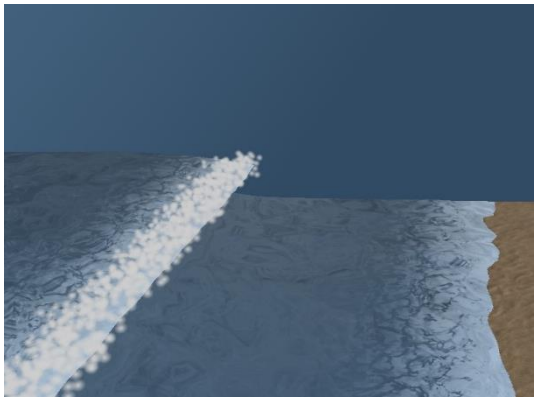

**Figure 28: Particles spawned along a wave mesh**

### 2.10.5 Wetting the Sand

When a shoreline advances and water flows over sand the sand gets wet and retains the water when the shoreline recedes. Although the sand is not part of the water it has the potential to affect the overall appearance of the simulation. Since the moving shoreline is a prominent feature of the simulation, it is important to ensure the sand reflects this. The sand is given the appearance of wetness by darkening the output value in the pixel shader. Over time the sand dries and will return to maximum wetness when it is next covered by the shoreline.

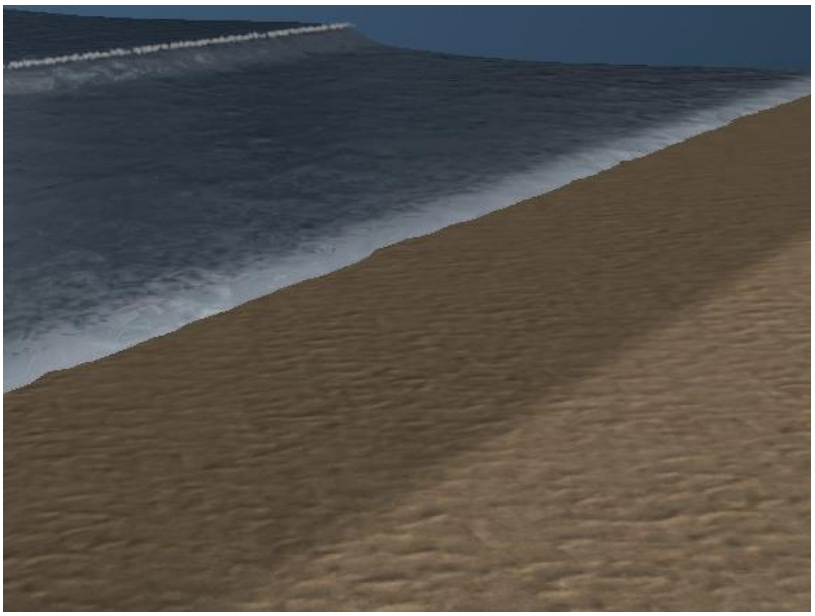

**Figure 29: Sand is made wet by the moving shoreline**

# Chapter 3. Results and Discussion

This chapter will now evaluate the simulation with regards to its performance and determine how visually realistic it is in order to address the research question:

*"How realistic can a simulation of shoreline water be in a real-time application which also incorporates breaking waves?"*

## 3.1 Performance Analysis

The application's performance was analysed using high resolution profiling code from Bremner (2010). This allowed specific components of the simulation to be profiled to determine where the simulation spends most of its time. All testing was carried out using a release mode build of the application on a laptop with the following specifications:

**Table 1: Specifications of the laptop used for testing**

| | |
|---|---|
| CPU | Intel Core i5-3230M (3 MB Cache, 2.60 GHz) |
| RAM | 6 GB |
| GPU | Intel HD Graphics 4000 |
| OS | Microsoft Windows 10, 64-bit |

The performance of the simulation is important if it is to be used within video games. As explained earlier the simulation should "be at least around 3–20 times faster" (Kellomäki 2012, p.11) than real-time due to the other systems in video games which require processing power.

The simulation was profiled with a height field of dimensions 250x200, where 200 is the length of the height field along the axis parallel to the shoreline, and 250 is the length of the height field along the axis perpendicular to the shoreline. These values were appropriately chosen so that the height field extends into the distance, imitating a scene within a video game. The results shown in Table 2 were produced by running the simulation over a period of twenty seconds. The 'number of executions' column in Table 2 shows how many times each component of the simulation was executed over the test period. Some components of the simulation will always be executed every frame, whereas wave meshes and splash particles will not always exist so are not updated.

**Table 2: Results from profiling components of the simulation**

| Component of the simulation | Number of executions over 1245 frames | Time in milliseconds per execution |
|---|---|---|
| Complete Simulation | 1245 | 1.683 |
| Update height field and normals | 1245 | 0.987 |
| Solving shallow-water equations | 1245 | 0.559 |
| Updating wave meshes | 429 | 0.044 |
| Update ripples and normals | 1245 | 0.025 |
| Update the sands wetness | 1245 | 0.012 |
| Update the splash particles | 521 | 0.009 |

As shown in Table 2, the whole simulation takes on average just 1.683 milliseconds to execute. In the context of a gaming application we can consider real-time to be a speed of 60fps, where one frame lasts for 16.667 milliseconds. With an execution time of 1.683 milliseconds, the simulation is processed 9.9x faster than real-time, using only 10% of the processing power available in a single frame.

### 3.1.1 Frame Rate

Having a fast execution time does not guarantee that the rendering of the simulation will be fast. The uncapped frame rate of the application was tested with the same 250x200 sized height field over a period of twenty seconds. The results are presented in Table 3 below, showing the difference between a windowed and full screen version of the application. Included in the application is an animated sky which compliments the simulation, however this will have a negative effect on the frame rate.

**Table 3: Frame rate tested in windowed and full screen modes**

| Mode | Frames per second | Milliseconds per frame |
|---|---|---|
| Windowed (800x600) | 475.737 | 2.102 |
| Full Screen (1600x900) | 148.082 | 6.753 |

### 3.1.2 Effect of the Height Field's Size on Simulation Time

From Table 2 it is found that the most computationally expensive components of the simulation are numerically solving the 1D shallow-water equations and updating the vertices of the height field (including the normals). These computational times are directly affected by the size of the height field (see Figures 30 and 31).

The effect that the length of the shoreline has on the performance of the simulation can be seen in Figure 30. Since the shallow-water equations are only solved along the axis perpendicular to the shoreline, extending the length of the shoreline has no effect on the time taken to solve the shallow-water equations. By increasing the length of the shoreline, the number of vertices within the height field is also increased, thereby increasing the overall time taken to update the height field.

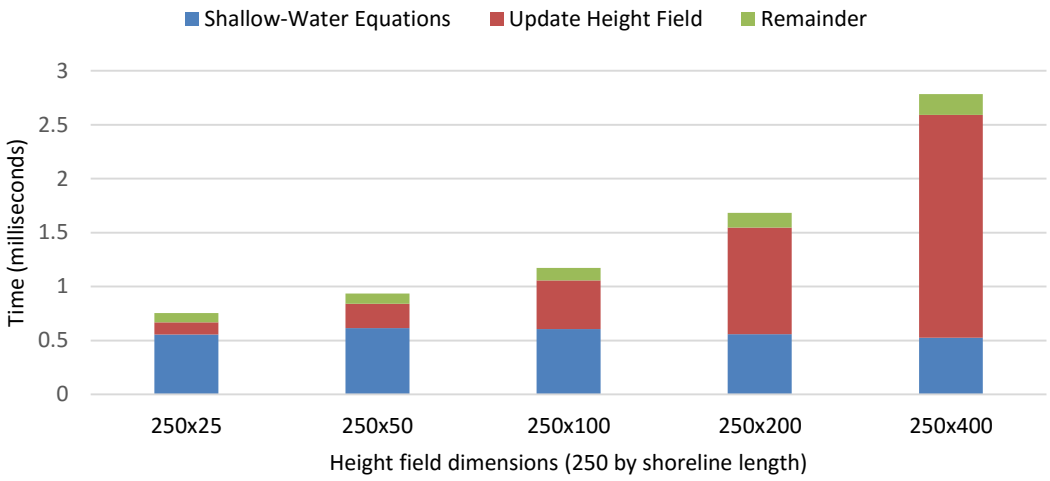


**Figure 30: Effect of shoreline length on simulation time**

Extending the height field in the direction perpendicular to the shoreline has a greater impact on the simulation time (see Figure 31). This is due to the increased number of points to solve the shallow-water equations over.

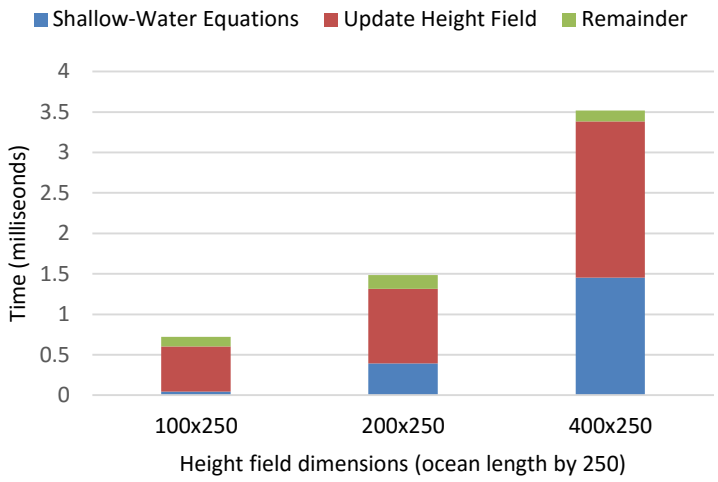


**Figure 31: Effect of ocean length on simulation time**

By comparing the results from Figures 30 and 31 (see Figure 32) it is observed that, for height fields of equal areas, the simulation will run faster where the length of the height field along the shoreline is longer.

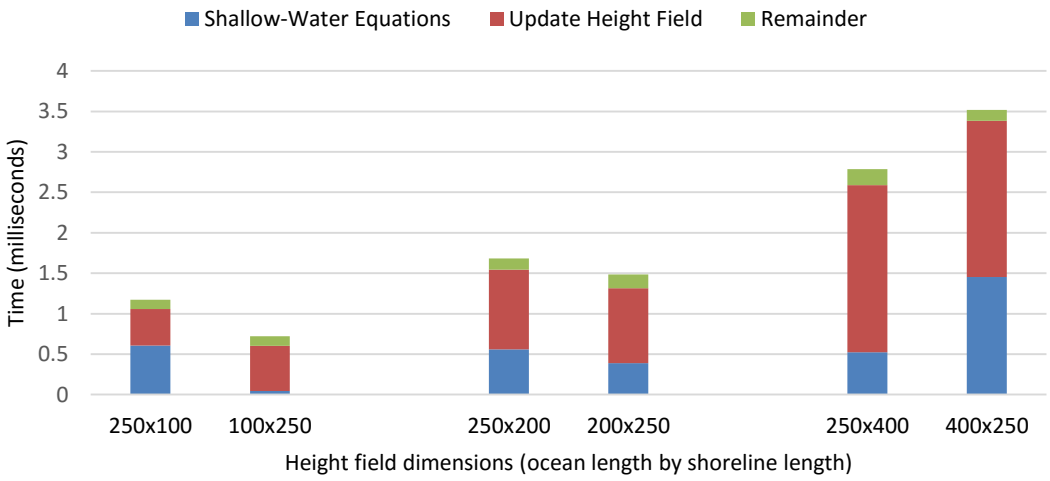


**Figure 32: Comparison of results from Figures 30 and 31**

### 3.1.3 Stability of the Simulation

In order to best demonstrate the simulation, the user has the ability to modify the heights of the propagated waves, ranging from a height of 0.1 to 0.5 metres. The simulation was left to run for a period of four hours at both the minimum and maximum wave heights. On both occasions the simulation remained stable, with no drop in water level.

## 3.2 Visual Analysis

With regards to the visual realism of the simulation, it is understood that the simulation should be believable and photorealistic if it is to be used within a modern video game.

### 3.2.1 Believability of the Simulation

Since the simulation uses the 1D shallow water equations, waves can only be produced along one axis. This restricts the topography of the sea bed and the land as it must remain consistent along the length of the shoreline. This of course limits the believability any shoreline water simulation can achieve, but the computational expense of using the 2D shallow-water equations would be too great for a real-time simulation.

Figure 33 depicts the progression of a wave breaking as it approaches the shoreline. Here the shoreline is seen to be receding as the wave approaches, accurately replicating the motion of water at a shoreline. The method used to produce the wave mesh, combined with the particle effects, successfully creates the impression of a breaking wave that would be seen at a beach. Although the wave breaks uniformly along its length, the simulation can be considered to be believable.

**Figure 33: Progression of a breaking wave**

### 3.2.2 Photorealism of the Simulation

All of the steps taken in an attempt to achieve photorealism within the simulation are necessary, with each improving its visual appearance. The use of particles to represent the breaking of waves significantly improves their photorealism as particles successfully distort the unnatural straightness of the waves. Particles also highlight the turbulence of crashing waves, with water splashing up from the surface upon their impact. Although particles visually add to the simulation they do not blend into the surface of the water as would be expected. They often appear detached and separate from the water (see Figure 33).

As seen in Figure 33 the foam produced at the shoreline successfully conveys its movement caused by the incoming waves. The methods used to create and render the foam provides detail and movement which bears a resemblance to foam observed at a beach (see Figure 34).

**Figure 34: (L) Photo of shoreline foam (R) Screenshot of simulated foam**

The surface detail on the water that was created from animated ripples and normal mapping, produces a surface similar to that seen in real life examples of the ocean (see Figures 35 and 36).

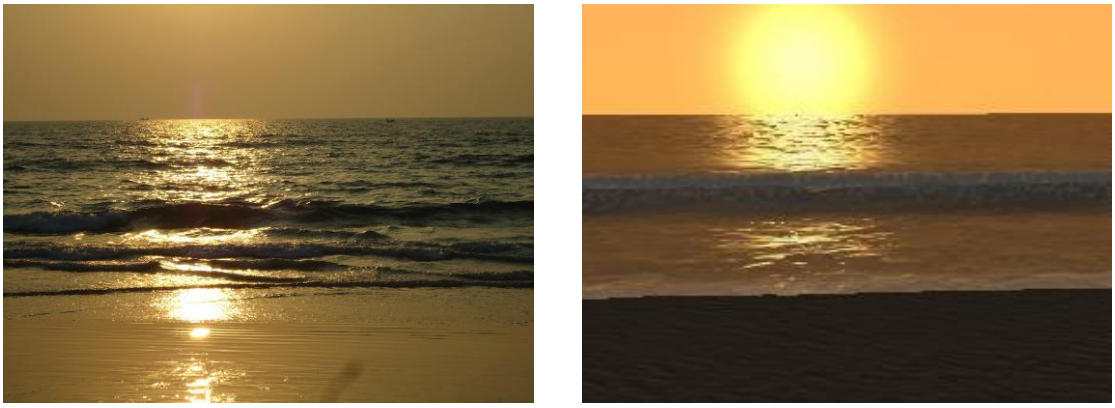

**Figure 35: (L) Photo of shoreline water (R) Screenshot of simulated water**

By reflecting the sky, including the clouds and the sun, the surface of the water is made photorealistic as the sunlight glistens across the water. This photorealism is maintained as the changing colour of the sky is reflected in the water (see Figure 36 and 37).

**Figure 36: The surface of the water reflecting the sun and the sky**

**Figure 37: The colour of the water changing as the sky changes**

From the results discussed above, the simulation can be considered somewhat photorealistic as there are areas where improvements could yet be made. Improvements to existing visual features can be incorporated with relative ease as the application provides a good foundation to build upon. When considering the photorealism of the simulation alongside its believability we can say that, overall the simulation is realistic and produces visually satisfying results.

# Chapter 4. Conclusions and Future Work

The project was undertaken with the aim to incorporate breaking waves into a real-time simulation of shoreline water, suitable for use in video games. After investigating the methods available for simulating shallow water and breaking waves, a simulation was then developed. The simulation produced was then evaluated to answer to the project's research question:

*"How realistic can a simulation of shoreline water be in a real-time application which also incorporates breaking waves?"*

Although not perfect, the simulation produced has been shown to be realistic whilst running faster than real-time. It represents shoreline water with breaking waves in a believable manner and also displays a good level of photorealism, accurately resembling the visual features of shoreline water. The most apparent problem with the simulation is the uniformity of breaking waves. Further investigation could look to resolve this and further improve the believability of the simulation, as it would more accurately reflect the behaviour of shoreline water.

To answer the research question, a simulation of shoreline water which also incorporates breaking waves has the potential to be fairly realistic at real-time. Over the course of the project, the four objectives outlined in the introduction have been met as well as achieving the two aims of the project and successfully answering the research question. With only a few minor improvements the current simulation could be made more realistic, making it suitable for use within a video game environment.

## 4.1 Future Work

Even though breaking waves have successfully been incorporated into a simulation of shoreline water, there is no variation within the waves other than their height. Waves seldom break along their whole length at once. The breaking of a wave usually takes place at single points along the wave, then spread out across the length of the wave. Given more time the generation of wave meshes and the spawning of particles could be adapted to make waves break in a more believable manner.

The simulation could be expanded to allow for more complex shoreline topographies, since the simulation is currently restricted to shorelines with a consistent topography along the length of the shoreline. This would require the use of the 2D shallow-water equations, greatly increasing the computational expense of the simulation. To reduce an incurred computational expense, the area manipulated by the 2D shallow-water equations could be limited to a smaller area close to the shoreline. The boundary values could then be manipulated from a second height field in the ocean, producing waves through methods such as the Sum of Sines Approximation (Finch 2007).

Visually there could always be room for improvement within the simulation. One visual aspect of shorelines which was not considered was the reflectivity of wet sand. This is only noticeable during a rising or setting sun (see Figure 35) and would be trivial to incorporate into the simulation.

# APPENDICIES

# APPENDIX A: Euler's Equations

The following two equations are known as the Euler equations (Bridson and Müller-Fischer 2007, p. 9). They are a simplification of the Navier-Stokes equations and model an ideal fluid which has no viscosity with a constant density:

$$\frac{\delta \boldsymbol{u}}{\delta t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\nabla \frac{p}{\rho} + \boldsymbol{g} \qquad (1)$$

$$\nabla \cdot \mathbf{u} = 0 \qquad (2)$$

Where $\boldsymbol{u}$ is the velocity, $p$ is the pressure, $\rho$ is the density of the fluid and $\boldsymbol{g}$ is the gravitational force. The first equation, known as the momentum equation, is essentially Newton's 2nd Law $F = ma$. The second equation, known as the incompressibility condition, states that the volume of the fluid remains constant throughout the simulation (Bridson and Müller-Fischer 2007, p. 7).

## APPENDIX B: Solving the 2D Dampened Wave Equation

The following equation is the two-dimensional wave equation:

$$\frac{\delta^2 \boldsymbol{h}}{\delta t^2} = C^2 \left( \frac{\delta^2 \boldsymbol{h}}{\delta x^2} + \frac{\delta^2 \boldsymbol{h}}{\delta y^2} \right) \qquad (1)$$

Where $\boldsymbol{h}$ is the height as a function of $x$, $y$ and $t$. $C$ is defined as a constant wave speed. This equation is a second order linear partial differential equation which can be solved by using the central difference formula and taking $\Delta x = \Delta y$ to give:

$$\frac{\boldsymbol{h}_{ij}^{n+1} - 2\boldsymbol{h}_{ij}^{n} + \boldsymbol{h}_{ij}^{n-1}}{\Delta t^2} = C^2 \frac{\left( \boldsymbol{h}_{i-1,j}^{n} + \boldsymbol{h}_{i+1,j}^{n} + \boldsymbol{h}_{i,j+1}^{n} + \boldsymbol{h}_{i,j-1}^{n} - 4\boldsymbol{h}_{ij}^{n} \right)}{\Delta x^2} \qquad (2)$$

When rearranged for $\boldsymbol{h}_{ij}^{n+1}$ this becomes:

$$\boldsymbol{h}_{ij}^{n+1} = 2\boldsymbol{h}_{ij}^{n} - \boldsymbol{h}_{ij}^{n-1} + K^2 \left( \boldsymbol{h}_{i-1,j}^{n} + \boldsymbol{h}_{i+1,j}^{n} + \boldsymbol{h}_{i,j+1}^{n} + \boldsymbol{h}_{i,j-1}^{n} - 4\boldsymbol{h}_{ij}^{n} \right) \qquad (3)$$

Where

$$K = \frac{C\Delta t}{\Delta x}$$

With a dampening term the two-dimensional wave equation is:

$$\frac{\delta^2 \boldsymbol{h}}{\delta t^2} = C^2 \left( \frac{\delta^2 \boldsymbol{h}}{\delta x^2} + \frac{\delta^2 \boldsymbol{h}}{\delta y^2} \right) + B \frac{\delta \boldsymbol{h}}{\delta t} \qquad (4)$$

Where $B$ is a constant for the amount of dampening applied. Using the central difference formula the dampening term becomes:

$$B \frac{\delta \boldsymbol{h}}{\delta t} = B \frac{\boldsymbol{h}_{ij}^{n+1} - \boldsymbol{h}_{ij}^{n-1}}{2\Delta t} \qquad (5)$$

By adding the dampening term (5) to the wave equation (2) and rearranging for $\boldsymbol{h}_{ij}^{n+1}$ we arrive at equation (6):

$$\boldsymbol{h}_{ij}^{n+1} = \frac{2\boldsymbol{h}_{ij}^{n} - \boldsymbol{h}_{ij}^{n-1} + K^2\left(\boldsymbol{h}_{i-1,j}^{n} + \boldsymbol{h}_{i+1,j}^{n} + \boldsymbol{h}_{i,j+1}^{n} + \boldsymbol{h}_{i,j-1}^{n} - 4\boldsymbol{h}_{ij}^{n}\right) + \frac{B\Delta t \boldsymbol{h}_{i}^{n-1}}{2}}{1 + \frac{B\Delta \mathrm{t}}{2}} \qquad (6)$$

For equation (6) to remain stable the following condition should be satisfied:

$$0 < \Delta t < \frac{\Delta x}{C\sqrt{2}}$$

# REFERENCES

# REFERENCES


*Assassin's Creed III*. 2012. [disk]. PC. Ubisoft.

*Assassin's Creed IV: Black Flag*. 2013. [disk]. PC. Ubisoft.

Bates, J.R. et al. 1993. A semi–Lagrangian approach to the shallow water equations. *6th Copper Mountain Conf. on Multigrid Methods, NASA Conference Publication 3224*. pp. 593–604.

Bim, J. 2001. *Fresnel Effect*. [online]. 3dRenderer.Com. Available from: http://www.3drender.com/glossary/fresneleffect.htm [Accessed 21 April 2016].

Bremner, J. 2010. *cRun Watch: Code Time Profiler*. [Source code]. Available from http://66.199.140.183/cgi-bin/ravenset.cgi/index [Accessed 24 April 2016].

Bridson, R. and Müller-Fischer, M. 2007. Fluid simulation: SIGGRAPH 2007 course notes. *ACM SIGGRAPH 2007 courses*. pp. 1–81.

Bruan, H., Raupp Musse, S. and Strube de Lima, D. 2010. A Model for Real Time Ocean Breaking Waves Animation. *2010 Brazilian Symposium on Games and Digital Entertainment*. pp. 19-24

CFD Online. 2012. *Navier-Stokes* Equations. [online]. Available from: http://www.cfd-online.com/Wiki/Navier-Stokes_equations [Accessed 26 November 2015].

Crespin, B. et al. 2011. A survey of ocean simulation and rendering techniques in computer graphics. *Computer Graphics Forum*. 30(1): pp. 43-60.

*Doom*. 1993. [disk]. MS-DOS. GT Interactive.

*The Elder Scrolls III: Morrowind*. 2002. [disk]. PC. Bethesda Softworks.

Finch, M. 2007. Chapter 1. Effective Water Simulation from Physical Models. In: R. Fernando, ed. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. [online]. 2004. Available from: http://http.developer.nvidia.com/GPUGems/gpugems_ch01.html [Accessed 27 October 2015].

*GoldenEye 007*. 1997. [cartridge]. Nintendo 64. Nintendo.

*Grand Theft Auto: San Andreas*. 2004. [disk]. PlayStation 2. Rockstar Games.

*Grand Theft Auto V*. 2014. [disk]. PlayStation 4. Rockstar Games.

Gross, M. et al. 2007. Real-time breaking waves for shallow water simulations. *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*. pp. 39-46.

Hestenes, M. and Stiefel, E. 1952. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*. 49 (6): pp. 409-436.

Khan, A.A. and Lai, W. 2014. Chapter four: One-dimensional conservation laws. *Modelling Shallow Water Flows Using the Discontinuous Galerkin Method*. Florida: CRC Press. 2014, pp. 62-64.

Kayne, F. 2014. *3D Water Effects: Here's What's Involved*. [online]. Available from: http://gamasutra.com/blogs/FrankKane/20140122/209052/3D_Water_Effects_Heres_Whats_Involved [Accessed 27 October 2015].

Kellomäki, T. 2012. Water simulation methods for games: a comparison. *MindTrek '12 Proceeding of the 16th International Academic MindTrek Conference*. pp. 10-14.

Kotzer, Z. 2015. *Rendering Realistic Water Is Still Game Development's Moby Dick*. [online]. Available from: http://motherboard.vice.com/read/rendering-realistic-water-is-still-game-developments-moby-dick [Accessed 26 November 2015].

Layton, A. and Van de Panne, M. 2002. A Numerically Efficient and Stable Algorithm for Animating Water Waves. *The Visual Computer*. 18(1): pp. 41-53.

*The Legend of Zelda: The Wind Waker*. 2002. [disk]. GameCube. Nintendo.

Randall, D.A. 2006. *The Shallow Water Equations*. [online]. Colorado State University. Available from: http://kiwi.atmos.colostate.edu/group/dave/pdf/ShallowWater.pdf [Accessed 16 March 2016].

Seymour, M. 2012. *Assassin's Creed III: The tech behind (or beneath) the action*. [online]. Available from: https://www.fxguide.com/featured/assassins-creed-iii-the-tech-behind-or-beneath-the-action/ [Accessed 7 March 2016].

*Shadow Warrior*. 1997. [disk]. MS-DOS. GT Interactive Software.

*Super Mario Sunshine*. 2002. [disk]. GameCube. Nintendo.

*Super Mario 64*. 1996. [disk]. Nintendo 64. Nintendo.

Surfing Waves. 2011?. *How Waves Break*. [online]. Available from: http://www.surfing-waves.com/waves/how_waves_break.htm [Accessed 19 April 2016].

NOAA. 2016. *Oceans & Coasts*. [online]. Available from: http://www.noaa.gov/oceans-coasts [Accessed 5 March 2016].

Rath, R. 2014. *Why Games are Terrible at Water*. [online]. Available from: http://www.escapistmagazine.com/articles/view/video-games/columns/criticalintel/11840-Why-Games-are-Terrible-at-Water [Accessed 26 November 2015].

Robert, A. 1981. A semi-Lagrangian, semi-implicit numerical integration scheme for the primitive meteorological equations. *Atmos-Oceans*. 19: pp. 35–46.

*Wave Race 64*. 1996. [cartridge]. Nintendo 64. Nintendo.

Wong, M. et al. 2013. A cell-integrated semi-Lagrangian semi-implicit shallow-water model (CSLAM-SW) with conservative and consistent transport. *Monthly Weather Review*. 141: pp. 2545-2560.

# BIBLIOGRAPHY

# BIBLIOGRAPHY


Burden, R.L. and Faires, J.D. 2001. Chapter 7.5: The Conjugate Gradient Method. *Numerical Analysis*. 7th ed. Brooks/Cole Publishing Company pp. 464-478.

Gourlay, M.J. 2014. *Fluid Simulation for Video Games (part 1)*. [online]. Available from: https://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-1 [Accessed 7 March 2016].

Kontaxis, C. 2013. *Fluid Simulation for Computer Graphics*. [Unpublished Master's thesis]. Utrecht University.

Robinson, K. 2015. *Here's How Nintendo Created the Genius Water Physics in Super Mario Sunshine*. [online]. Available from: http://www.gamnesia.com/news/heres-how-nintendo-created-the-genius-water-physics-in-super-mario-sunshine [Accessed 6 March 2015].

Sherrod, A. 2008. Chapter 14: Water Rendering. *Game Graphics Programming*. Course Technology. 2008. pp. 511-524.