

České vysoké učení technické v Praze  
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

## ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Daniel Mikeš**

Studijní program: Otevřená informatika  
Obor: Počítačová grafika a interakce

Název tématu: **Simulace vodní hladiny pomocí vlnových částic**

Pokyny pro vypracování:

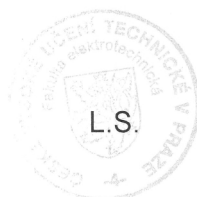
Prostudujte metody simulace a vykreslování vodní hladiny. Zaměřte se zejména na metody simulace vodní hladiny umožňující řešení interakce s objekty v reálném čase. Implementujte metodu Wave Particles [1], která reprezentuje vlnu pomocí částic uspořádaných ve dvourozměrné mřížce a umožňuje simulovat interakci vodní hladiny s plovoucími objekty. Implementace bude realizována na GPU s využitím rozhraní OpenGL. Funkčnost implementace otestujte jak v rámci simulace rozlehlých vodních ploch, tak simulace vodní hladiny omezené rozlohy. Navrhněte optimalizace časově nejnáročnějších fází simulace a vykreslování. Důkladně vyhodnoťte časovou náročnost implementace v závislosti na počtu částic a počtu objektů v rámci nejméně třech různých scénářů: otevřené moře, mořský břeh, bazén. Zhodnoťte kvalitu simulace porovnáním s video sekvencemi z reálného prostředí.

Seznam odborné literatury:

- [1] Cem Yuksel, Donald H. House, John Keyser. Wave Particles. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007), 26, 3, 2007.
- [2] Michael B. Nielsen, Andreas Söderström, and Robert Bridson. Synthesizing waves from animated height fields. ACM Transactions on Graphics, 32, 1, 2013.
- [3] E. Darles, B. Crespin, D. Ghazanfarpour, J.C. Gonzato. A Survey of Ocean Simulation and Rendering Techniques in Computer Graphics. Computer Graphics Forum, 30, 1, 2011.

Vedoucí: doc. Jiří Bittner Ing., Ph.D.

Platnost zadání: do konce letního semestru 2015/2016



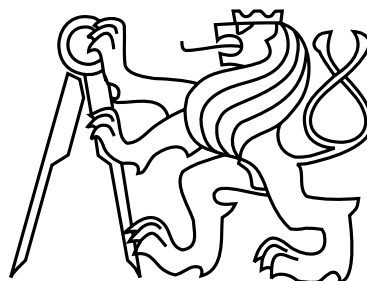
prof. Ing. Jiří Zára, CSc.  
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.  
děkan

V Praze dne 4. 11. 2014



Czech Technical University in Prague  
Faculty of Electrical Engineering  
Department of Computer Graphics and Interaction



Masters thesis

**Real-time water surface simulation with user interaction  
using Wave particles**

*Bc. Daniel Mikeš*

Supervisor: Ing. Jiří Bittner, Ph.D.

Study Programme: Open Informatics, Masters program

Field of Study: Computer graphics

January 5, 2015



## Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.

I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague on January 5, 2015

.....



# Abstract

When rendering large bodies of water in real-time an efficient method is required to model water waves. This thesis describes a method for real-time interactive generation of such waves. We use the *wave particle* method to describe wave propagation in a fluid medium. The method allows to simulate interactions of water with general shaped rigid bodies in real-time. We present a GPU implementation of the method and show results in scenarios such as open ocean waters or pools with water boundaries. Finally we compare the tested result to the wave propagation in a real life.

# Abstrakt

Při vykreslování velkých vodních ploch v reálném čase, je zapotřebí efektivní metody k reprezentaci tohoto jevu. Tato diplomová práce popisuje metodu pro interaktivní simulaci vodních ploch pomocí metody *vlnových částic* v reálném čase. Tato metoda umožňuje simulovat interakci vody s pevnými tělesy obecných tvarů. Tato práce představuje GPU implementaci této metody a demonstruje výsledky ve scénářích jako např. otevřené mořské hladiny nebo bazény s ohraničením. Závěrem porovnává výsledky testování se šířením vln v reálném životě.





# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Subject of this thesis . . . . .	2
1.3	Thesis structure . . . . .	2
<b>2</b>	<b>Theoretical background</b>	<b>3</b>
2.1	Volume based fluid simulations . . . . .	3
2.1.1	Navier-Stokes equations . . . . .	4
2.1.2	Lagrangian approach . . . . .	4
2.1.3	Eulerian approach . . . . .	6
2.1.4	Hybrid methods . . . . .	6
2.2	Heightfield representation . . . . .	7
2.2.1	Spatial domain . . . . .	7
2.2.1.1	Gerstner Waves . . . . .	8
2.2.2	Spectral domain . . . . .	9
2.3	Dynamic properties of water body . . . . .	10
2.3.1	Wave propagation . . . . .	10
2.3.2	Water depth . . . . .	11
2.3.3	Breaking waves . . . . .	13
2.3.4	Continuous flow . . . . .	13
2.3.5	Water wakes . . . . .	14
2.4	Optical properties of water . . . . .	14
2.4.1	Water caustics . . . . .	14
2.4.2	Reflection and Refraction . . . . .	15
2.4.3	Godrays . . . . .	15
2.4.4	Fresnel term . . . . .	15
2.4.5	Water colour . . . . .	16
2.4.6	Whitecaps . . . . .	17
2.5	Fluid interaction . . . . .	17
2.5.1	Volumetric simulations . . . . .	17
2.5.2	Heightfield based simulations . . . . .	18
<b>3</b>	<b>Water Simulation with Wave Particles</b>	<b>19</b>
3.1	Motivation . . . . .	19
3.2	Wave equation . . . . .	19

3.2.1	1D Wave equation . . . . .	20
3.2.2	Solution to the wave equation . . . . .	20
3.2.3	2D Wave equation . . . . .	21
3.3	Wave particles . . . . .	22
3.3.1	Wave particles for 1D wave . . . . .	22
3.3.2	Wave particles for 2D wave . . . . .	23
3.3.3	Radial deviation function . . . . .	25
3.3.4	Longitudinal waves . . . . .	26
3.3.5	Wave particle properties . . . . .	27
3.3.6	Wave particle subdivision . . . . .	29
3.3.6.1	Subdivision criterion . . . . .	30
3.3.6.2	Creating new wave particles . . . . .	31
3.3.7	Water boundary . . . . .	33
3.4	Object to fluid interaction . . . . .	36
3.4.1	Wave-object collision . . . . .	36
3.4.2	Wave generation . . . . .	37
3.4.2.1	Wave position . . . . .	37
3.4.2.2	Wave propagation . . . . .	38
3.4.2.3	Wave volume . . . . .	39
3.4.3	Wave particle generation . . . . .	39
3.5	Fluid to object interaction . . . . .	40
3.5.1	Buoyancy force . . . . .	40
3.5.2	Dynamic forces . . . . .	41
<b>4</b>	<b>Used technologies</b> . . . . .	<b>43</b>
4.1	Other dependencies . . . . .	43
4.2	OpenGL library . . . . .	43
4.3	GLSL . . . . .	44
<b>5</b>	<b>Implementation</b> . . . . .	<b>45</b>
5.1	Application structure . . . . .	45
5.2	Wave particle method . . . . .	46
5.2.1	Buffer size . . . . .	47
5.2.2	Particle data structure . . . . .	47
5.2.3	Wave particle simulation . . . . .	49
5.2.3.1	Propagation routine . . . . .	49
5.2.3.2	Subdivision and delete routine . . . . .	50
5.2.3.3	Particle generation routine . . . . .	51
5.2.4	Wave particle reflection . . . . .	52
5.3	Filter particles . . . . .	53
5.4	Water to object interaction . . . . .	54
5.4.1	Buoyancy force . . . . .	55
5.4.2	Drag and lift force . . . . .	55
5.4.3	Water-object collision . . . . .	56
5.4.4	Parallel reduction . . . . .	56
5.5	Object to water interaction . . . . .	56

5.6 Floating object . . . . .	58
<b>6 Results</b>	<b>61</b>
6.1 Simulation scenarios . . . . .	61
6.2 Quality testing . . . . .	61
6.3 Performance testing . . . . .	61
6.4 Volume testing . . . . .	63
<b>7 Conclusion</b>	<b>71</b>
7.1 Future work . . . . .	71
<b>A List of abbreviations</b>	<b>85</b>
<b>B User manual</b>	<b>87</b>
B.1 Controls . . . . .	87
<b>C DVD content</b>	<b>89</b>



# Chapter 1

## Introduction

One of the goals of computer graphics is to capture and reproduce real life phenomena as truly as possible. Water has a crucial role in life and people are naturally fascinated with it. Faithfully water is a very common object in computer graphics. Due to high complexity of dynamical and optical properties of fluids, water rendering is still an open challenge.

Displaying open water scenes is not necessarily about the optical properties of water and the rendering part of the process. Phenomena as wind or motion of an object in the water volume can create waves, foam, water sprinkles and other effects which need to be described in order to achieve realistic results.

In computer graphics there is always a trade-off in the manner of computational cost and realism of final image. Today's methods for modelling and rendering physically correct water are offering high level of realism. On the other hand these methods are also very complex and cannot be simply applied in real-time graphics. Thus it is important to distinguish between real-time and offline methods.

Since the goal is to obtain images of real scenes it is necessary to take interaction with the environment into account. The major drawback of offline approaches is the fact that it is not possible to perform interaction with a user on the fly. On the other hand there are different requirements on computational time and the quality of the result in different simulation scenarios. When we are trying to achieve realistic results in terms of fluid simulation, representation of fluid is also important part of the application.

### 1.1 Motivation

In real-time applications such as video games it is crucial that all the computations are fast enough to be computed in a plausible frame rate, so both the rendering and the simulation stages should be fast. In general lots of computationally complex tasks can be pre-computed and then used later on. Even measurements of real life phenomena can be used to speed up the calculations. The problem is that some applications may also require user interaction with the object in the water volume which can cause some unpredicted calculations meaning that pre-computation is not possible in this scenario.

For reasons mentioned above we usually need to settle for approximate solutions, which means both the rendering and the simulation step is often not physically correct but offers reasonable results.

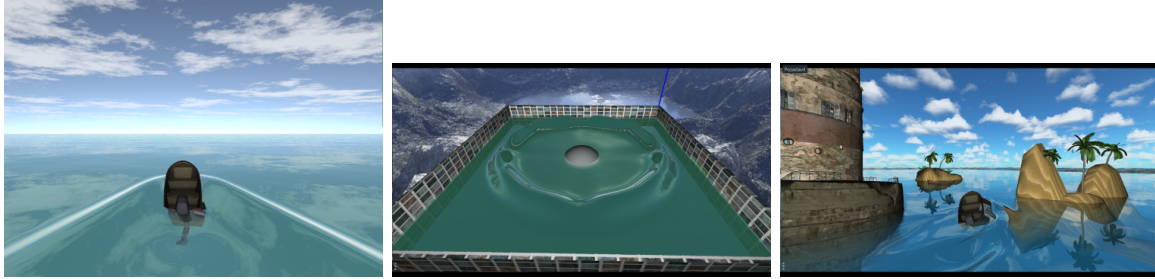


Figure 1.1: Results of our application.

## 1.2 Subject of this thesis

This thesis investigates the Wave particle method for water wave generation introduced by Yuksel et al. in 2007 [YHK07] and the research done in the field of real-time methods for wave generation. The presented method describes a real-time simulation of water surface with user interactions. In contrast to the original article we will implement this method fully on the GPU (the original article describes CPU implementation in some parts).

We will mainly aim at open ocean scenes with no boundaries, but we will also focus on different scenarios. For example closed water areas such as swimming pools where waves can reflect off of a solid boundary. Another type of scenes are near-shore areas, where breaking waves can be found. We will also implement method for water surface rendering and discuss various effects which arise in connection with fluid dynamics. Different approaches used in both real-time and offline methods will be discussed.

## 1.3 Thesis structure

Thesis is divided into three main parts. In chapter 2 we will discuss water representation in computer graphics simulations. In the same chapter we depict main phenomena which occur in real life water bodies. Chapter 3 describes the wave particle method in detail. Chapter 4 briefly summarizes used technologies. Consequently chapter 5 describes implementations steps. And finally chapter 6 shows the results of our application.

## Chapter 2

# Theoretical background

In this chapter we will describe the theory behind the water wave distribution and the physics of fluids. We will also introduce main problems in this field.

Simulations are usually an imitation of a real life phenomenon. Most simulations tend to be simplified or approximated forms of some physical law or measurement. It is important to understand the physical meaning of the phenomenon to develop a suitable simulation.

There are two distinct approaches of how the fluids are represented in computer graphics. We will describe features, advantages, and drawbacks of these methods.

### 2.1 Volume based fluid simulations

Volume representation is based on physical nature of fluids where every piece of fluid volume interacts with its surroundings. This approach tries to imitate all the properties of real life water phenomenon such as water flow (velocity propagation in the volume), surface tension or heat propagation. In computer graphics water volumes have to be discretized in order to be stored in a memory. Different techniques use a different way of discretization.

Eventually results obtained from a volume based simulation are presented. Rendering of these results can be done by creating a boundary representation of the volume data. This step is only a presentation of the results and it is not part of the simulation process. In contrast with boundary representation, volume based methods capture information in each cell of the volume. Volume data can be directly rendered by a ray marching method, where no surfaces are created.

Due to the high complexity of 3D fluid volumes, these methods are often used only in offline computations. Even in offline methods resources are limited, which is the reason why volume based methods are usually used for detailed but limited simulations such as shallow waters or in differently restricted water bodies e.g. liquid in a bottle, swimming pool etc.

Volume base methods are also suitable for depicting dynamic effect like splashes, water drops or breaking waves.

### 2.1.1 Navier-Stokes equations

Navier-Stokes equations (NSE) describe physics of many phenomenon we can observe in a real life. NSE are used mainly for computing flow in the fluid volume. It has a practical use in aerodynamics, weather forecasting and oceanography.

$$\nabla \mathbf{u} = 0 \quad (2.1)$$

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{\nabla p}{\rho} + \mu \frac{\nabla^2}{\rho} + \frac{f}{\rho} \quad (2.2)$$

Where  $\mathbf{u} = (u_x, u_y, u_z)$  is the fluid velocity,  $t$  is the time,  $p$  is the pressure,  $\rho$  is the fluid density,  $\mu$  is the fluid viscosity,  $f$  is a external force (e.g. gravity),  $\nabla$  represents gradient and is defined as  $\nabla \mathbf{u} = (\frac{\partial u_x}{\partial t}, \frac{\partial u_y}{\partial t}, \frac{\partial u_z}{\partial t})$ , and  $\nabla^2$  is the Laplace operator.

A usual case is to capture incompressible flow which can simplify NSE with a constraint of constant density as shown in equation 2.1. Nevertheless compressible fluids can also be captured. In terms of modelling water flow for computer graphics there is no need for using the compressible fluid schema in a standard simulation environment. Phenomena such as fluid in a strong mechanical vibration requires using compressible schema e.g. air propagation in an environment with a sound source.

In theory Navier-Stokes equations are still not fully understand. The problem of the existence of a smoothness solution is even one of the Millennium problems. In addition NSE is a non-linear equation due to convective acceleration, which is in contrast to local acceleration not dependent on time but on the position of the fluid *element*. Convective acceleration arises from movement of a larger set of elements (e.g. molecules) of a fluid. In equation 2.2 the term  $\mathbf{u} \cdot \nabla \mathbf{u}$  is the representation of this acceleration.

Note that the solution of the Navier-Stokes equation is not the position of the volume, but its velocity. In order to solve the equation we need to ensure the boundary conditions in a form of velocity interacting with the fluid, e.g. wind above the water.

Due to high complexity of its solution, NSE has to be discretized. There are two different application of NSE that differs in a way of discretization: Eulerian and Lagrangian approaches. Quantities of Navier-Stokes equation are velocity, viscosity, pressure and density. These quantities (if not constant) have to be stored in each fluid element of the discrete domain.

### 2.1.2 Lagrangian approach

In Lagrangian approach, discretization is performed spatially<sup>1</sup> on the mass of the simulated fluid. Fluid element in Lagrangian method can be seen as a particle representing some part of the water mass similarly to molecules in real life. Because all the particles quantities are variable we formulate a function of particle identification  $p_i$ .

$$\mathbf{v} = f(p_i, t)$$

---

<sup>1</sup>Relatively to the fluid volume.



Lagrangian methods are meshfree which means that there is no need for a grid that holds the fluid as shown in figure 2.1.

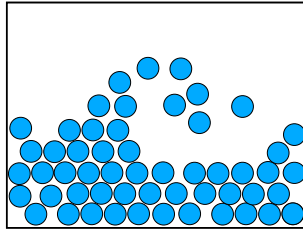


Figure 2.1: Illustration of Lagrangian fluid element forming 2D wave.

We can classify methods from the Lagrangian class by the quantity that the particles hold. Momentum particles<sup>2</sup> are used in a method called Smoothed-particle hydrodynamics. Another approach is to use vortons which are special particles with circular motion and refer to the fluid vorticity. Besides water simulation, this method is used mainly for simulation scenarios such as smoke, fire, and explosions.

**Smoothed-particle hydrodynamics** Smoothed-particles hydrodynamics is a fluid simulation technique which uses a set of particles. Contribution of the particle quantities are weighted and summed together based on the spatial distance from each other. There are also two approaches in the way of smoothing quantities between particles. One way is to search for actual neighbours and then linearly interpolate the value. The other way is to perform smoothing base on a kernel function and the smoothing distance of a particle.

This equation quantifies an arbitrary quantity  $A$  (e.g. velocity, density) of fluid particle in the point  $x$  in the volume:

$$A(\mathbf{x}) = \sum_j m_j \frac{A_j}{\rho_j} W(|x - x_j|, h_j), \quad (2.3)$$

where  $x_j$  the position,  $m_j$  is the mass and  $A_j$  is the chosen quantity of  $j$ -th particle.  $\rho_j$  is the density associated with the  $j$ -th particle and  $h_j$  is its smoothing distance, and  $W$  represents the smoothing kernel function.

The choice of the smoothing kernel function is arbitrary but it should fulfil certain quality. It should be a smooth, non-negative, and differentiable function with compact support<sup>3</sup>. The compact support is important in order to achieve profitable performance, because the broader the support is, the more particles is being weighted.

Onderik et al. [OCv13] presented a SPH method improvement with small scale details such as splashes and foam. They described a technique for improved surface reconstruction by density normalization resulting into more compact regions of thin water volume. They also presented a method for faster spatial particle sorting.

<sup>2</sup>Description of mass and velocity in the volume.

<sup>3</sup>The function is non-zero at a finite range.

### 2.1.3 Eulerian approach

Eulerian approach discretizes the domain into a spatial<sup>4</sup> grid. In Eulerian approach fluid elements can be seen as a voxel, which contains quantities of the volume. In each cell, fluid velocity is evaluated and the volume moves from one voxel to another. In contrast to Lagrangian class positions are fixed in each cell and volume quantities are dependent variable. In other words chosen quantity can be formulated as a function of position  $\mathbf{x}$  and time  $t$  as shown in figure 2.2.

$$\mathbf{v} = f(\mathbf{x}, t)$$

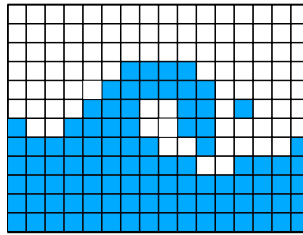


Figure 2.2: Illustration of Eulerian fluid cell forming 2D wave.

Advantage of Lagrangian method is that the mass conservation is computed implicitly compared to Eulerian method where extra computations are necessary. On the other hand Lagrangian methods need large number of particles in order to develop same resolution as the Eulerian methods.

Thurey et al. [TRS06] use a combination of 2D and 3D Eulerian grid. 2D grid is used for representing wave propagation on the surface, 3D grid is used for a limited area centred at the position of interacting object and it captures drops of water and other more dynamic changes in the local area.

Due to the fact that Eulerian methods usually need a large grid for plausible simulation results, there is an effort to use appropriate data structures. Some of these data structures are discussed in course notes from Bridson et. al [BMF07].

Irving et al. [IGLF06] described a method which uses Eulerian approach to simulate 3D water volume with a grid cell reduction. On the water surface where most details occur the grid has a high granularity while in the lower parts of the fluid grid cells are grouped together. Grid cells are group into vertical stripes with the same horizontal size as the other cells. This means that the cells have different size in horizontal dimension in order to simplify the computations.

### 2.1.4 Hybrid methods

There are methods which do not fit into strict classification. One of them is method presented by Chentanez et al. [CM10] which combines SPH with an Eulerian method creating a shallow water simulation in real-time. They aim simulation scenarios like raging river waters, waterfalls and breaking waves.

---

<sup>4</sup>Relatively to the fluid container.

Raveendran et al. [RWT11] propose a method for preserving uniform particle density in SPH. They implement a coarse grid which helps distributing values to the particles instantaneously.

## 2.2 Heightfield representation

An effective way to increase simulation performance of large bodies of water is to reduce the problem from three to two dimensions. In contrast to volume approaches, which store information needed for the simulation in 3D space, heightfield methods represent only a 2D surface. Due to this reduction dynamic effects of the water volume are ignored and only the information about boundary is present.

The surface can then be deformed in the vertical direction in order to model a wave on the surface to create an illusion of waves propagating through water body. Representation of a scalar data in a 2D grid is called height field and it is described as a 2D function  $h(x, y)$ . A problem arises when we are trying to assign more values to same input parameters conflicting with definition of a single value function, which is the case of breaking waves or other high dynamic effects of the water surface.

Simulations using heightfield representation are usually rendered using boundary representation of the water surface. Advantage of this approach is the aforementioned domain reduction and the fact that rasterization process is well supported on GPUs at the hardware level and thus it is very fast. The drawback of this class of methods is that we do not have the information about the object's inner part.

### 2.2.1 Spatial domain

The goal of the spatial domain methods is to describe water surface geometry in terms of a set of sinusoidal functions. Motion of the waves is achieved by changing the phase of individual source sinusoidals. Every wave sinusoidal function represents the contribution the surface deformation. This exploits the superposition principle mentioned in 3.2.2.

Similar idea originally came from Max [Max81]. He describes the height function  $y = h(x, z, t)$  as follows:

$$d_v(\mathbf{x}, t) = y_0 + \sum_{i=1}^N A_i \cdot \cos(\mathbf{k}_i \cdot \mathbf{x} - \omega_i t + \varphi_i), \quad (2.4)$$

where  $\mathbf{x} = (x, z)$  is the position in the horizontal plane,  $y_0$  is the initial water level height,  $N$  number of wave components,  $A_i$  is the amplitude,  $\mathbf{k} = (k_{i_x}, k_{i_z})$  is the wave vector which is a horizontal vector pointing in the direction of wave propagation,  $\omega_i$  is wave speed (also pulsation), and  $\varphi_i$  is the phase of the  $i$ -th wave component.

The wave vector is related to the as:

$$k = \frac{2\pi}{\lambda}, \quad (2.5)$$

where  $k = |\mathbf{k}|$  is the wave number and  $\lambda$  is the spatial frequency of the wave component. In real fluids the  $\omega$  is somehow related to the wavelength  $\lambda$ . The origin of the relation will be discussed in 2.3.2.

Since then there was lot of applications of this formula. For instance Chen et al. [CLW<sup>+</sup>07] used this approach using shaders and bump mapping to create small ripples water surface.

Advantage of this approach is that we have control over all wave components and we can change the waves in the whole spectrum.

### 2.2.1.1 Gerstner Waves

Gerstner waves is a model of wave in the spatial domain, which depicts waves with sharper peaks and flatter troughs as compared to the model proposed by Max described in section 2.2.1.

The effect of blowing wind in combination of existing waves on the water surface can result into a sharpening of the wave peaks. This is a common effect and can be observed mainly on deep waters or oceans where circular wave motion is most significant. Circular motion will be discussed in 2.3.1.

The Gerstner wave model consists of a vertical deviation component  $h$  and horizontal deviation component  $d_h$ . Even in real life we can observe similar phenomenon of horizontal waves as shown in 2.3.1. Figure 2.3 depicts the effect of the vertical deviation on the wave.

$$d_v(\mathbf{x}, t) = y_0 + A \cos(\mathbf{k} \cdot \mathbf{x} - \omega t), \quad (2.6)$$

$$d_h(\mathbf{x}, t) = \mathbf{x} - \frac{\mathbf{k}}{k} A \sin(\mathbf{k} \cdot \mathbf{x} + \omega t), \quad (2.7)$$

where,  $d_h$  ( $d_v$ ) is the horizontal (respectively vertical) deviation function,  $\mathbf{x} = (x, z)$  is the position on the horizontal plane,  $\mathbf{k}$  is the wave vector,  $\omega$  is the wave speed,  $t$  is the time, and  $A$  is the amplitude.

Note that the parameters must be set with caution in terms of water wave simulation, because horizontal deviation may cause undesirable self intersections. For  $kA < 1$  (figure 2.3 left) we can observe that the Gerstner model produces plausible results without self intersection. A loop is formed on the wave crest if  $kA > 1$  as shown in figure 2.3 right.

We can apply equation 2.7 to a set of wave components similarly to 2.4:

$$d_v(\mathbf{x}, t) = y_0 + \sum_{i=1}^N A_i \cos(\mathbf{k}_i \cdot \mathbf{x}_i - \omega_i t + \varphi_i), \quad (2.8)$$

$$d_h(\mathbf{x}, t) = \mathbf{x} - \sum_{i=1}^N \frac{\mathbf{k}_i}{k_i} A_i \sin(\mathbf{k}_i \cdot \mathbf{x}_i + \omega_i t + \varphi_i), \quad (2.9)$$

After summing the horizontal deviation for all wave components  $i$  (as shown in figure 2.4), self intersection may be present but not visible for the current phase shift configuration. The choice of parameters which yield  $\sum_{i=1}^N k_i A_i < 1$  will ensure no intersection.

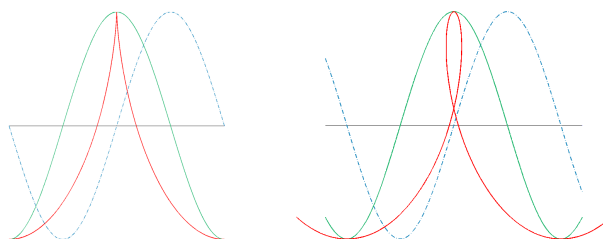


Figure 2.3: Model of Gerstner wave in 2D. Plausible result (left), the case of exaggerated self intersection artefact (right). Green line shows original wave without vertical deviation, sharp red line shows a wave after vertical deviation, and the blue dotted line shows the vertical deviation. We put the vertical deviation in one image although the horizontal value is the deviation in vertical axis.

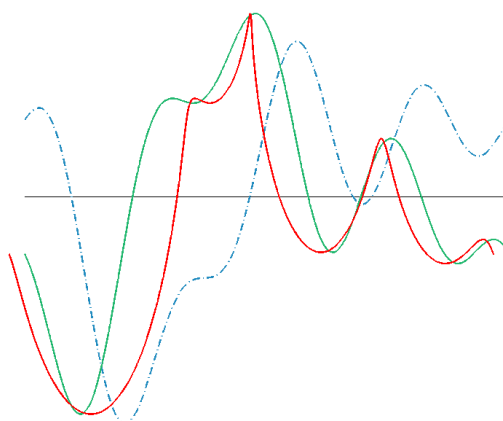


Figure 2.4: Model of Gerstner wave in 2D composed out of 3 components. Original wave (green), the final wave (red), and the vertical deviation (blue).

### 2.2.2 Spectral domain

The idea behind spectral domain approaches is to represent water surface from the knowledge of spectral distribution of waves. Spectral distribution of real water can be measured by specialized sensors used for oceanographic research. Spectral domain will be then decomposed to the spatial representation using inverse Fast Fourier transform [DCGG11].

Tessendorf [Tes01] presented a method for simulating ocean water. He uses spectral domain to describe the water surface. The input spectrum is computed by Gaussian pseudo-random generator, which generates data that are somehow similar to the frequency domain of the real waters. We can write down a Fourier transform based representation of the heightfield:

$$d_v(\mathbf{x}, t) = \sum_{\mathbf{k}} \tilde{d}_v(\mathbf{k}, t) e^{i\mathbf{k}\mathbf{x}}, \quad (2.10)$$

where  $\mathbf{x}$  is the spatial position in time  $t$ ,  $\tilde{d}_v$  is the vertical deviation function  $d_v$  in frequency domain, and  $\mathbf{k}$  is the wave vector (wave frequency for each direction of the wave).

Spectral domain methods can produce waves of high level of realism, although the result depends on the choice of the spectrum. Tessendorf also described a Philips spectrum, which is a different spectrum suitable for deep water scenes with wind driven waves.

This method produces periodic waves on a patch with a limited horizontal size. Advantage is that we can reuse this patch by repeating it infinitely creating seamless tiles. On the other hand one can observe some linearities when the period (patch size) is not large enough. Additional noise can be used to conceal this undesirable effect. Disadvantage of spectral domain approaches is that the whole patch must be computed at once. This makes it hard to locally modify part of the heightfield. Resulting into difficult adoption of some extensions such as adaptive shallow water wave modification or user interaction with the generated water.

## 2.3 Dynamic properties of water body

In this section we will describe some of the main phenomena which occur in real life water bodies and are necessary part of a convincing simulation.

### 2.3.1 Wave propagation

Waves can be perceived at the interface between two different media and are mainly created by wind changes, underwater flows or by some other mechanical force. The wave is defined by its amplitude, wave length and phase speed. Other parameters like gravity, density and surface tension also change the behaviour of the wave propagation in a medium.

Waves do not occur only on the water surface. Turbulence in the water appear in the whole mass and the nearer to the surface the more influence does the turbulence have. This phenomenon is hard to observe in clean water volumes but is necessary for proper fluid flow in more complex simulations.

Typical ocean waves with relatively high wavelengths and small amplitudes are sometimes called surface gravity waves. The name comes from the fact that these kind of waves carry a lot of mass leading to the wave being affected by gravity more than a small waves. Swell is a series of waves with low frequencies, which arise from a strong consistent wind blow. Due to the low frequency these waves can travel very long distances without dispersing. On the other hand there are small waves created locally on the water surface also called ripples or capillary waves. Ripples are formed mostly by the wind and fluid surface tension. The influence of gravity on these waves is almost neglectable.

**Circular water motion** When observing a water wave we usually think of it as a set of water particles are going through the volume from the wave source to the place where the wave disperses. This is a special case called *soliton* or the *wave of translation*. In contrast to that, particles which are part of the wave crest are not necessarily the same particles which create the wave at the end of its lifetime. Instead water particles move in a circular trajectory according to the wave parameters. Figure 2.5 demonstrates this effect in

a deep water environment compared to shallow water figure 2.6. Note that the ratio between amplitude and wavelength affects the distance that the fluid particle travels. Figure 2.6 also explains the creation of breaking waves in shallow water environments.

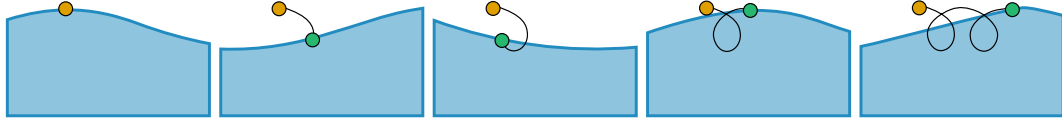


Figure 2.5: Frames taken from a water particle motion animation in deep waters. A water particle in a propagating wave in time (from left to right). The orange point is the initial position of the water particle. Current water particle position is marked by green point.

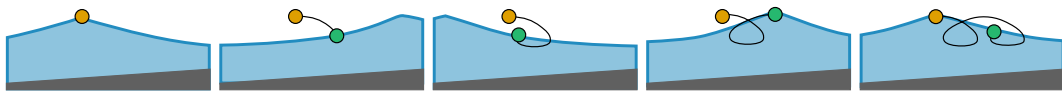


Figure 2.6: Frames taken from a water particle motion animation in shallow waters.

**Horizontal wave** Circular motion of the water particles leads to the observation that water waves consist of two components: traverse waves (vertical deviation) and longitudinal waves (horizontal deviation). Figure 2.7 demonstrates these components.



Figure 2.7: Illustration of the horizontal wave. Vertical component (left), horizontal component (centre), compounded wave (right). Courtesy of [YHK07].

**Wave diffraction** Wave diffraction is a effect which occur when a wave encounters an obstacle or a slit in an obstacle where a portion of the wave is suspended by the obstacle while the other portion of the wave travels further in the propagation direction. After a part of wave is separated it can be seen that the other part not only propagates in the origin direction but also diffracts into other directions. Huygens–Fresnel principle explains this effect so that each wave is also a source of new waves at the same time. Figure 2.8 illustrates this phenomenon. Consequence of the wave diffraction effect is that the waves can be observed even if the direct path from the source to the receiver is occluded by an obstacle.

### 2.3.2 Water depth

Due to the convective acceleration as the depth in the fluid increases the velocity and the radius of the particle motion decreases. As can be seen in figure 2.9 particles located deeper in the water appear to have lower frequency which is the result of Doppler effect since the particle on the surface level travels longer distance.

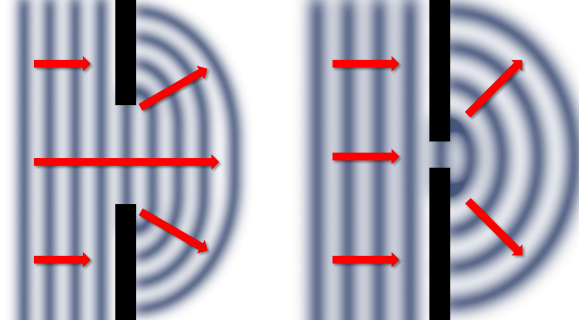


Figure 2.8: Wave diffraction after the wave passes the slit. Courtesy of [YHK07].

Figure 2.9 shows the difference between circular motion in deep water environment compared to the shallow waters. We consider shallow water if the distance from the water level to the ground becomes less than approximately one half of the wavelength.

In shallow waters the circular motion is disturbed by the ground and the particles is pushed up. This leads to the sharpening wave crests and eventually into wave breaking.

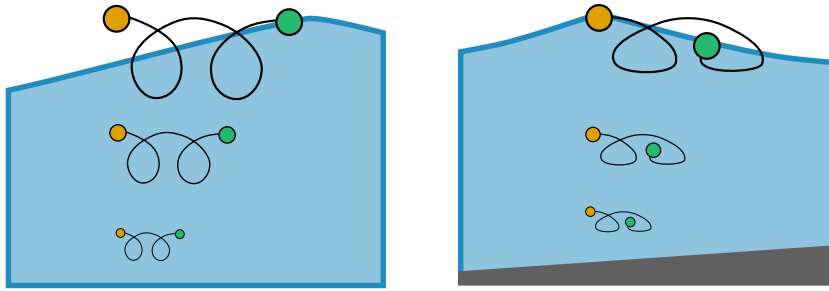


Figure 2.9: Circular motion of under water flows in deep water (left) compared to the shallow waters (right). The motion of the water particles located deeper inside the water is reduced exponentially.

**Dispersion relation** Dispersion relation refers to the dispersion of wave parameter while the wave is propagating in a medium. This term is rather general and it can describe relation of arbitrary parameter. For water waves there is a interesting relation between the wave vector and its magnitude. In other word we describe a relationship between wave frequency and the propagation speed. For deep waters the relations is

$$\omega(k)^2 = gk, \quad (2.11)$$

where  $\omega$  is the phase speed,  $k$  is the wave number, and  $g$  is the gravitational acceleration. Indirectly we can see that the phase speed varies as the square root of the wavelength.

For shallow water scenarios phase speed dispersion relation is described as

$$\omega(k)^2 = gk \tanh(kh), \quad (2.12)$$



where  $h$  is the height of the water column from the ground to the water surface.

In contrast to gravitation waves capillary waves neglect the effect of the gravity and its dispersion relation describes the influence of surface tension.

$$\omega(k)^2 = \frac{\sigma}{\rho + \sigma} |k|^3, \quad (2.13)$$

where  $\sigma$  is the surface tension, and  $\rho$  is the density.

One of the consequences of the equation 2.11 is that a wave created by a wave superposition are not necessarily stable in time because each wave with a different frequency has a different phase speed. This means that waves with higher frequencies have higher phase speed and will separate from the distributing wave front.

### 2.3.3 Breaking waves

Breaking waves occur when the turbulations of underwater flows interact with the water ground and pull the mass up creating a large amplitude. Similar result may be caused by a strong wind. While the amplitude is increasing the wave crest steepens and a relatively high and narrow wave becomes unstable in the motion leading to the collapse of the wave crest.

An offline method using Eulerian approach to simulate breaking waves is described by Mihalef et. al [MMS04]. They have created a tool for interactive modelling of the breaking wave called *slice method*. Artist can modify the breaking wave front by modifying a 2D slice (solving Navier-Stokes equations in two dimensions) of the 3D wave.

Thurey et al. [TMFSG07] presented a real-time method for breaking waves extraction from animated heightfield. First they locate regions with steep wave fronts and mark them with line which will generate particles which are connected together. These particle are used for creating the surface patch on the breaking wave an extension to the basic heightfield formulation. Particles which hit the surface of the basic heightfield are absorbed and can be converted into a foam or water drops.

### 2.3.4 Continuous flow

Continuous water flow is a simulation scenario where water is continuously moving and does not become stationary. For example raging rivers are one of most discussed scenarios. This environment is hard to represent using only heightfield simulations because the depth and steepness of the ground change rapidly resulting into dynamic velocity change in the water volume.

Eulerian methods described in 2.1.3 are suitable for simulation scenario like rivers. Although there are some methods for river flow simulation using heightfield representation. These methods add detail to surface in order to create an illusion of a flow and assume flat ground rather than focusing rivers with more complex profile.

Yu et al. [YNBH09] depicted a method for real-time animation of rivers. They capture a local velocity field of the river flow considering obstacles in form of 2D convex hull seen from above. Then particles are uniformly distributed into visible area and are used for texture advection, which adds detail to the surface according to the velocity flow.

Yu et al. [YNS11] tackled simulation of local surface deformation caused by an obstacle in the river flow. They analyse this phenomenon on real life scenes and address shock wave effect. They find a starting point in front of the obstacle in the direction of the flow. At this point a surface is created in order to display ripples which are surrounding the obstacle.

### 2.3.5 Water wakes

Water wakes are special waves created by turbulence left by object moving in the water volume. The shape of the wakes depends on objects hydrodynamical properties, its speed, and size. Doppler effect can be observed on the water wakes.

## 2.4 Optical properties of water

The rendering equation is an essential equation in computer graphics. It puts incoming and outgoing radiance in relation. It is derived from the energy conservation.

$$L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) \cos\theta_i d\omega_i, \quad (2.14)$$

where  $L_o$  ( $L_i$ ) is the outgoing (respectively incoming) radiance,  $L_e$  is the radiance emission,  $\mathbf{x}$  is the point of the surface,  $f_r$  represent the bidirectional reflectance distribution function (BRDF),  $\Omega$  is sphere of all radial angle,  $\omega_o = (\theta_o, \varphi_o)$  is the direction of outgoing ray,  $\omega_i = (\theta_i, \varphi_i)$  is the negative<sup>5</sup> direction of incoming ray,  $\theta$  is the polar angle,  $\varphi$  is the azimuthal angle of the spherical coordinates, and  $\lambda$  is a light wavelength.

Sometimes wavelength parameter is neglected in order to reduce the dimension of the rendering equation.

The choice of BRDF depends on the optical properties of the surface which we are trying to describe. Ross et al. [RDP05] presented a BRDF function for ocean rendering. They aim mainly sun glint and sky reflections.

Rendering equation can be solved with different methods which are part of a rendering algorithm. For example Monte Carlo methods are used by Path tracing algorithm [Hav].

In this section we will briefly describe some of the visual effects related to water simulation. There are also approximate methods which despite physical correctness try to maximize perception of the simulated phenomenon.

### 2.4.1 Water caustics

In optics, a caustic is an effect which arises from refracting or reflecting light from a curved object. When a set of light rays is reflected from a curved surface and the reflected rays are projected onto another surface. This rounding causes the rays not being distributed uniformly resulting into light spots, where more reflected rays met. Dark spots are similarly created in the place where a small number of rays intersected.

---

<sup>5</sup>Put into the same system as the outgoing ray.

Yuksel et al. [YK09] presented a real-time method for caustic extraction from an existing height field. They assume a flat ground underneath the water. Contribution to the caustic field is computed by the water depth and normal per each pixel and then filtered in a local neighbourhood to produce caustic texture, which can be mapped onto the surface.

### 2.4.2 Reflection and Refraction

The rendering equation mentioned above describes the relation between incoming and outgoing radiance. The contribution of the reflection is then described by BRDF and by the angle  $\theta_i$ .

By reflection we denote the sum of incoming radiance which was captured from the hemisphere above the surface and the following is valid  $\theta_t \in (-\frac{\pi}{2}, \frac{\pi}{2})$ . Refraction is denoted as a sum of incoming radiance which arrived from the hemisphere under the surface.

Note that the requirements for the BRDF function can change according to the dynamic properties of the water. For instance windy oceans with breaking waves have different reflectivity compared to the calm water in a mountain lake.

### 2.4.3 Godrays

Godrays is an effect which occurs when small particles are spread in an optical medium. Godrays can be observed on the sky in cloudy weather or in a slightly polluted water. It is also a consequence of light scattering.

### 2.4.4 Fresnel term

When light crosses the boundary between two optical media, part of the light intensity transmits into the new medium and part of it reflects according to the index of refraction of both media. The ratio of reflected and refracted intensity (reflectivity) is depicted by the Fresnel term.

Fresnel described the reflectivity for horizontally and vertically polarized light:

$$R_v = \left| \frac{\eta_1 \cos\theta_i - \eta_2 \cos\theta_t}{\eta_1 \cos\theta_i + \eta_2 \cos\theta_t} \right| \quad (2.15)$$

$$R_h = \left| \frac{\eta_1 \cos\theta_t - \eta_2 \cos\theta_i}{\eta_1 \cos\theta_t + \eta_2 \cos\theta_i} \right|, \quad (2.16)$$

where  $\eta_1$  and  $\eta_2$  are indices of refraction for the original and new medium respectively,  $\theta_i$  and  $\theta_t$  is the direction of the incoming (respectively transmitting) light ray,  $R_v$  and  $R_h$  is the reflectivity component of the vertically and horizontally polarized light.

The relation between angles  $\theta_i$  and  $\theta_t$  can be expressed by Snell's law:

$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_2}{\eta_1} \quad (2.17)$$

The circularly polarized light can be expressed as mean value:

$$R = \frac{R_v + R_h}{2}. \quad (2.18)$$

Figure 2.10 (left) shows individual components.

Fresnel term can be also approximated by Schlick's model. Comparison of this approximation is shown in figure 2.10 (right).

$$R_0 = \frac{\eta_1 - \eta_2}{\eta_1 + \eta_2} \quad (2.19)$$

$$R = R_0 + (1 - R_0)(1 - \cos\theta_t)^5 \quad (2.20)$$

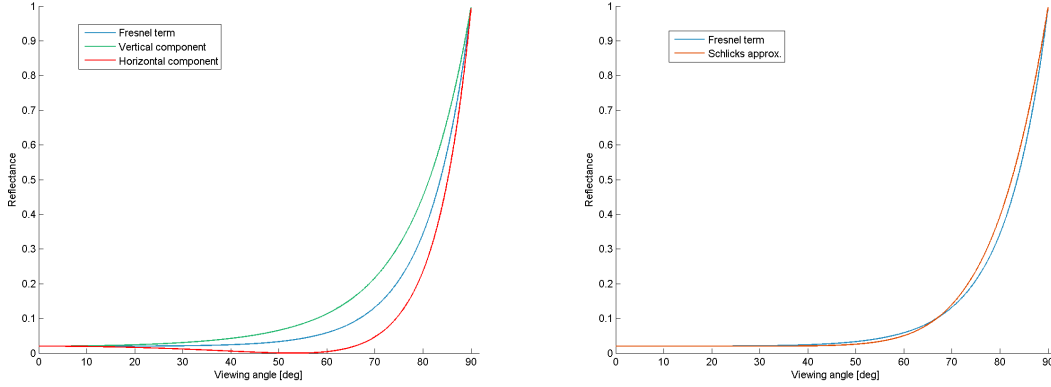


Figure 2.10: Reflectivity for a smooth water. Difference between the horizontally and vertically polarized component (left). Comparison of Schlick's approximation (right).

### 2.4.5 Water colour

Colour of water is highly dependent on the thickness of observed water volume. Small volumes of water seem to be colourless while large bodies of water are perceived in a shade of blue. This phenomenon is created by light scattering in the fluid volume. In optically participating media such as clouds or polluted water we have to take light scattering and absorption into account.

Rayleigh scattering arises when the light ray is scattered off of a molecule in an optical medium. Therefore, it depends on the wavelength of the light and the size of the particle (molecule). The radiant intensity  $I$  of the light being scatter from one particle can be described as [You81]:

$$I = I_0 \frac{8\pi^4 \alpha^2}{\lambda^4 R^2} (1 + \cos^2\theta), \quad (2.21)$$

where  $I_0$  is the initial intensity,  $\alpha$  is the polarizability of the molecule,  $\lambda$  is the wavelength,  $R$  is the distance from particle, and  $\theta$  is the scattering angle.

The important idea from equation 2.21 is that the scattering ratio can be simplified as

$$I = I_0 \frac{1}{\lambda^4}.$$

We can see from this relation that the higher the wavelength the higher the dispersion is. This means that only lower wavelengths of the colour spectrum will pass through and thus the colour of water is perceived blue.

Note that this applies for clean water. Impurities in the volume may cause dispersion of different colour spectrum.

### 2.4.6 Whitecaps

In high dynamic water environment whitecaps may emerge. This phenomenon is created by small bubbles and foam which can emerge on the water surface on account of wind. Dupuy et al. [DB12] presented a method for creating and rendering real-time ocean whitecaps. They use Choppy wave model [Tes01] to represent the ocean waves. They localize changes in the vertical deviation of the surface in order to find waves which may break and create a whitecap. These locations are then shaded by a per-pixel Lambertian shading model.

## 2.5 Fluid interaction

When modelling fluid-solid interaction we need to represent the obstructions in a way in which the simulation can understand these boundaries. Different simulation approaches need different data representation of the obstruction. These differences are described in the next section.

We distinguish one-way from two-way interaction scheme. We speak of one-way scheme when there is either only water to object or object to water interaction.

### 2.5.1 Volumetric simulations

In Eulerian framework the solid boundary can be easily depicted as a special voxel value. We can then recognize which voxels contain fluid and which represent a boundary or solid object. Unfortunately this leads to irregularly shaped fluid containers. Feldman et al. [FOK05] describe a Eulerian fluid simulation method for use on general tetrahedral meshes.

Solid boundary treatment (SBT) is an algorithm class for ensuring fluid to solid interaction in Lagrangian framework. Since particles in Lagrangian methods are not bounded to any particular position, there is a need for restraining the simulation space. Similarly to Eulerian methods one solution is to introduce a new type of particle which does not change its position and embodies a part of a solid object boundary. This particle is called repulsive particle and is responsible for creating repulsive force on the fluid particles. Schechter et al. [SB12] describe another type of particle - the ghost particles. Ghost particles are able to accumulate velocity which is then used for object motion in the fluid volume.

Treuille et al. [TLP06] present a method for faster object to fluid interaction. They use a 3D semi-Lagrangian approach for simulating the fluid. They optimize the object interaction

with precalculation of the *boundary bases*. A boundary base represents the velocity variation in the object volume. Linear combination of boundary bases from different time steps is used to cancel out velocities of particles that would otherwise pass through the object.

### 2.5.2 Heightfield based simulations

Tessendorf [Tes04] describes the *iWave* method for simple two-way fluid to object interaction in height field representation. He proposes using a convolution scheme instead of spectral method to simulate the water surface waves. This decision has been made considering low adaptability of spectral methods for interactive scenarios. This convolution is used for distributing the value changes into corresponding neighbours according to the velocity. He demonstrates his method on a CPU implementation using 2D regular grids which have the same resolution as the heightfield grid and are aligned with each other. One of these grids is used for generating new impulses interacting with the water surface generating new waves. There is also a grid representing solid object flattened in two dimensions which interacts with the water surface. The interaction is done simply by masking the height value in the corresponding height field.

The filter kernel is as follows:

$$G(x, n) = \sum_{i=-n}^n x_i^2 e^{-\sigma x_i^2} J_0(|x - x_i|), \quad (2.22)$$

where  $x$  is the grid point,  $n$  is the kernel size,  $x_i$  is the neighbour point of  $x$  in the distance of  $n$ ,  $\sigma$  similarly to Gaussian filter represents the standard deviation,  $J_0$  is the Bessel function. Note that the result should be normalized.

Chou et al. [CF07] described a simple method for ocean simulation with one-way interaction between the water surface and rigid bodies. Water simulation was based on heightfield using a form of Gerstner waves. They refer to the vertices of the heightfield grid as particles which store the information about horizontal and vertical surface deviation and represent the water flow. The flow is then used for water to object interaction.

## Chapter 3

# Water Simulation with Wave Particles

In this chapter we will depict the main ideas of the algorithm of Water simulation with wave particles presented by Yuksel et al. [YHK07].

### 3.1 Motivation

Phenomena occurring on real life water have been discussed in chapter 2. Successful simulation should maximize the resemblance to these effects. One of the most evident phenomenon of the water simulation is the wave generation and propagation. More importantly waves created by user interaction are essential to credibility of the simulation, because the user pays more attention to the system reaction if he or she is part of the simulation. In addition the human eye is more sensitive to the correctness of the motion than for instance water reflections which are hard to comprehend for the bare eye.

Besides the simulation credibility another important aspect is the simulation performance in order to obtain interactive frame rates. This is true for a general simulation case. In our water simulation it is also important that we have control over the data because we want to be able to localize and react to the generated waves. Flexibility to adopt a new simulation scenario is also important property. We also want our interaction waves to be separable from the environmental waves.

### 3.2 Wave equation

We begin by depicting wave equation which describes the propagation principle of electromagnetic, acoustic and other mechanical waves. Wave equation is hyperbolic partial differential equation and represents partial second derivative of space with respect to time.

$$\frac{1}{v^2} \frac{\partial^2 d_v(\mathbf{x}, t)}{\partial t^2} = \nabla^2 d_v(\mathbf{x}, t) , \quad (3.1)$$

where  $v$  is wave propagation velocity,  $d_v(\mathbf{x}, t)$  represent the vertical deviation function,  $\mathbf{x} = (x_1, x_2, \dots, x_{n-1})$  is a spatial vector in  $n$ -dimensional domain,  $t$  is time and  $\nabla^2$  is the Laplacian operator.

Equation 3.1 describes propagation of the wave with speed  $v$  in a space defined by the scalar function  $h$ . Function  $h$  will be later on denoted as *vertical deviation function*. Vertical deviation function takes spatial vector of  $n-1$  dimension and time variable as an input. Note that we can replace time in equation 3.1 with *horizontal* shift in the domain and transform this equation into second derivative of space with respect to position of the wave. We will show 1D and 2D Wave equation in detail and describe how that corresponds to the Laplacian operator.

Laplacian operator is a convolution function which investigates gradient differential in local neighbourhood. It can also be seen as a second order spatial derivative. Note that this convolution is energy conserving.

### 3.2.1 1D Wave equation

In 2-dimensional domain (change of one variable with respect to the other one) a 1-dimensional wave can be observed. We can substitute  $x = \mathbf{x}$  and see it as scalar in 1-dimensional wave. To create such 1D wave equation we simply take equation 3.1 and substitute  $y = d_v(x, t)$ .

$$\frac{1}{v^2} \frac{\partial^2 y}{\partial t^2} = \nabla^2 y \quad (3.2)$$

In equation 3.2 we understand  $\nabla^2$  as 1D Laplace operator, which ultimately represents second derivative of vertical deviation function  $y$  with respect to position of the wave  $x$ . Thus we can substitute this derivative to the formula.

$$\frac{1}{v^2} \frac{\partial^2 y}{\partial t^2} = \frac{\partial^2 y}{\partial x^2} \quad (3.3)$$

### 3.2.2 Solution to the wave equation

In order to numerically solve equation 3.3 and describe a concrete wave we have to state initial condition for this differential equation. Wave particle method solves wave equation analytically in a discrete domain.

If we assume constant velocity  $v$ , we can see that a simple solution to the wave equation is a single wave propagating in both direction as shown in figure ???. Referring to the equation 3.3 it satisfies the condition that change of  $y$  according to  $t$  can be expressed in the form of change of  $y$  according to  $\mathbf{x}$ . Therefore, any solution is eventually in the form

$$d_v(\mathbf{x}, t) = f(\mathbf{x} + t\mathbf{v}) + g(\mathbf{x} - t\mathbf{v}), \quad (3.4)$$

where  $f$  and  $g$  are arbitrary vertical deviation functions. It represents two waves that are approaching each other in opposite direction. We can also fix one the function, remove the shift and write it in a integral form to cover the whole function support:



$$\int_{-\infty}^{\infty} d_v(x, t) dx = \int_{-\infty}^{\infty} f(x) + g(t - x) dx. \quad (3.5)$$

Equation 3.5 is a generalized form of the convolution theorem which explains the application of Laplacian operator. It also means that solution to the equation 3.2 can be obtained via convolution.

Functions  $f$  and  $g$  in equation 3.4 satisfy an important property of a travelling function

$$f(\mathbf{x}, t) = f_s(\mathbf{x} + t\mathbf{v}), \quad (3.6)$$

where  $f$  is the function defined in space and time and  $f_s$  is the same function defined only in space.

Note that even a solution to the wave equation does not result into a physically correct wave with all the phenomena we discussed in 2.3. On the other hand we can find a wave configuration which will not satisfy the wave equation and still produce a relatively realistic results.

**Superposition principle** Superposition principle states that a response caused by more stimuli is equal to the sum of the responses caused by the individual stimulus. Consequence of this is that each wave can be described as a sum of other waves shown in figure 3.1. Principle of superposition is the basic idea used in Fourier transform.

Since wave equation is linear it satisfies the superposition principle. This applies to the waves in the manner that if two independent waves satisfy wave equation then sum of these waves is also a solution to the wave equation.

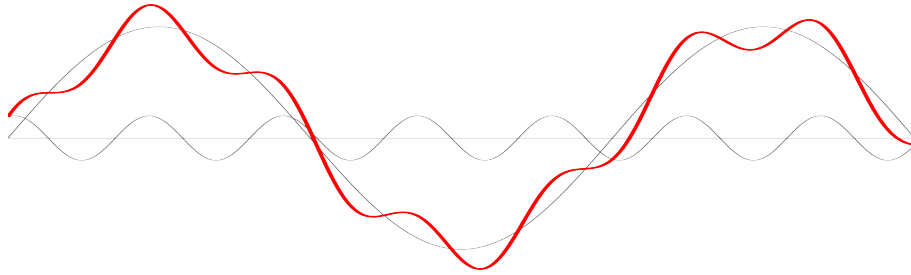


Figure 3.1: Modelling a wave using superposition principle. The red wave is the superposition of the blue and green one.

Due to the superposition principle we can separate waves created by user interaction and the ambient waves created by another unlocalized source.

### 3.2.3 2D Wave equation

In order to describe water waves in 3D world we need to come up with a 2D wave equation. Composing this equation is quite straight-forward. In equation 3.1 we substitute  $z = d_v(\mathbf{x}, t)$  as  $z$  is the last coordinate in 3D domain and  $\mathbf{x} = (x, y)$ .

$$\frac{1}{v^2} \frac{\partial^2 z}{\partial t^2} = \frac{\partial^2 z}{\partial x^2} \frac{\partial^2 z}{\partial y^2} \quad (3.7)$$

### 3.3 Wave particles

Wave particle method uses a particle system for representing surface deviation. Each particle symbolizes a local vertical deviation function  $d_v$  mentioned in 3.2. Every wave particle has an information about its own position  $\mathbf{x}$  which is used for localizing the deviation function  $d_v(\mathbf{x}, t)$ . Note that particles are totally independent on each other and keep locally all the parameters needed for the distribution in the water. Other parameters will be discussed latter on.

Unlike Lagrangian methods wave particles move in a plane which is coplanar with the water surface. In other words the wave particles do not represent elements of the water mass, they represent only a deformation on water surface.

Set of local deviation function is then synthesized to the global deviation function

$$D_v(\mathbf{x}, t) = y_0 + \sum_{i \in P} d_{v_i}(\mathbf{x}, t), \quad (3.8)$$

where  $D_v$  is the global vertical deviation function of space  $\mathbf{x}$  and time  $t$ ,  $d_{v_i}(\mathbf{x}, t)$  is the local vertical deviation function of the  $i$ -th particle,  $y_0$  is water base level, and  $P$  is a set of all particles.

#### 3.3.1 Wave particles for 1D wave

We will again formulate a solution to the wave equation with the use of wave particles. In order to create a continuous wave front in 2D domain superposition principle is used to form one continuous wave front which consist of more particles. A wave created in such a way satisfies the wave equation because it is continuous, constant, and travelling function.

We formulate the vertical deviation function for the use of 1D wave

$$d_v(x, t) = A_i W(x - x_i(t)), \quad (3.9)$$

where  $x$  is the position on the x-axis,  $x_i(t)$  is the position of the  $i$ -th particle in time  $t$ ,  $A_i$  is the  $i$ -th wave particle amplitude, and  $W$  is the *waveform function*.

Waveform function is presented in account of weighting the contribution of deviation function of each particle with reference to its position.

$$W_i(u) = \frac{1}{2} \left( \cos \left( \frac{2\pi u}{l_i} \right) + 1 \right) \Pi \left( \frac{u}{l_i} \right), \quad (3.10)$$

where  $u$  is the distance from the position on the x-axis to the  $i$ -th particle,  $\Pi$  is the box function, and  $l_i$  is the length of the  $i$ -th particle. Although the particle length  $l_i$  is equal to the wavelength  $\lambda$  of the original sinusoid function we distinguish these terms to stress out

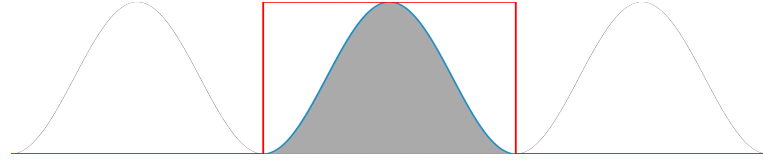


Figure 3.2: Illustration of box function. The grey sine wave is the original signal

that  $\lambda$  is used in the periodic function while  $l_i$  is the wavelength after it has been cut out by the box function  $\Pi$  and is not periodic again.

$$\Pi(x) = \begin{cases} 1, & -\frac{1}{2} \leq x \leq \frac{1}{2} \\ 0, & \text{otherwise} \end{cases} \quad (3.11)$$

The square function is used for restriction of the support due to periodic behaviour of the cosine function in two dimensional domain as shown in figure 3.2. The square function is used in order to localize the wave particle contribution.

Note that to choice of waveform function is arbitrary unless it satisfies certain conditions. These conditions are in fact general properties of a blending function similar to the one discussed in 2.1.2. Above that the shape of the function is beneficial in order to describe water surface waves.

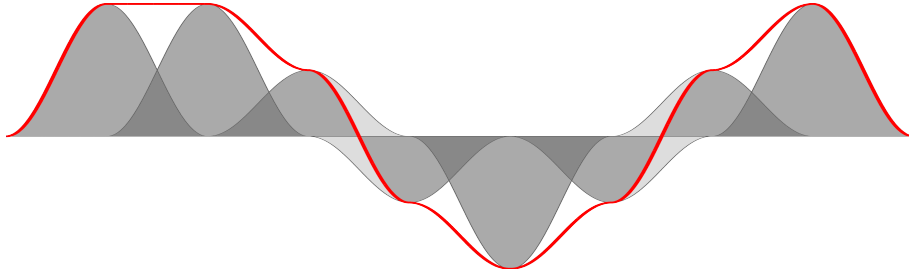


Figure 3.3: Superposition of the particle deviation function. The red line represents the square function, grey sine wave represents the original waveform function without and the blue is the

### 3.3.2 Wave particles for 2D wave

Propagation of the wave particle in three dimensional domain follows the same principle as in two dimensions but has an additional constraint. Unlike wave particles the previous case can propagate in only two directions, wave particles in 3D can be propagated into infinite number of directions onto 2D plane. This fact leads to a requirement of model which can handle continuous wave fronts in three dimensions. In contrast to the 1-dimensional wave, the direction of propagation is not the same as the direction of wave blending. The waveform function  $W$  was responsible for blending the waves in the direction of propagation.

In order to represent a continuous wave front we adopt an additional blending function  $B$ , which blends particles located next to each other. In a case of a linear wave front neighbouring

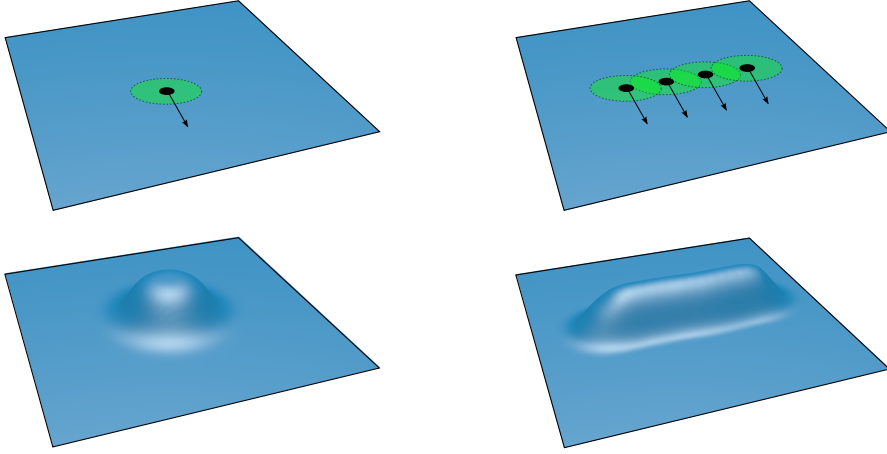


Figure 3.4: Illustration of the wave front generation. Top images show the position of the wave particles and its radius  $r_i$  (green circles). The bottom images show the global deviation function combined from local deviation function of these particles.

particles are located in the direction perpendicular to the propagation direction. Similarly to the 1D case, particles in the direction of the propagation are still blended via waveform function.

$$d_{v_i}(\mathbf{x}, t) = A_i W(\mathbf{v}_i \cdot \mathbf{u}) B(\mathbf{v}_i^\perp \cdot \mathbf{u}), \quad (3.12)$$

where  $\mathbf{u} = \mathbf{x} - \mathbf{x}_i$  is the distance to the  $i$ -th wave particle,  $\mathbf{v}_i$  is the propagation direction and  $B$  is the blending function.

In figure 3.4 relatively simple wave front is presented as a result of three wave particles. In practice the simulated wave fronts are more complex and are not necessarily linear.

In addition to linear wave fronts we recognize two other types of continuous wave fronts: *expanding* and *contracting*. These wave front types are formed similarly to the linear wave fronts but they propagate with a certain curvature  $\kappa$ .

Since the wave fronts are represented by particles which are discrete in space we assume that the part of the wave front curve which belongs to one wave particle is constant in one time step.

**Expanding wave fronts** Expanding wave front is a case of a wave front which arise when a wave is created in one point and distributes further in all directions.

Important property of expanding wave fronts is that the wave particle size is expanding in order to form continuous wave with the neighbouring particles while the diameter is enlarging. Note that due to the energy conservation amplitudes are also decreasing the further the wave fronts gets.

A perfectly expanding wave fronts can be observed when a round object is thrown in the liquid in the right angle and is represented in figure 3.5.

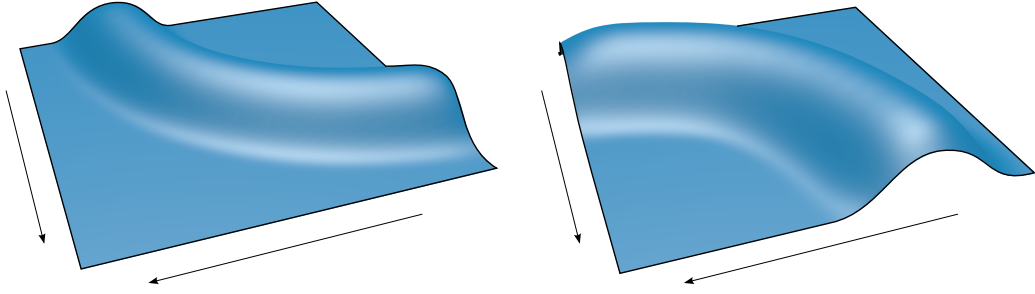


Figure 3.5: Illustration of wave fronts types: expanding wave front (left) and contracting wave front (right). Arrows represent the propagation direction.

**Contracting wave fronts** Similarly to the expanding wave front this is the case when the wave front propagates with a opposite curvature into one point. In this case wave particles are getting closer to each other creating a large compounded wave. Contracting wave fronts can be observed when the source of mechanical oscillation in the fluid has concave shape.

### 3.3.3 Radial deviation function

Since the wave particles can form arbitrarily shaped wave fronts we use another definition of vertical deviation function.

$$D_i(\mathbf{x}, t) = \frac{A_i}{2} \left( \cos \left( \frac{\pi |\mathbf{x} - \mathbf{x}_i(t)|}{r_i} \right) + 1 \right) \Pi' \left( \frac{|\mathbf{x} - \mathbf{x}_i(t)|}{2r_i} \right), \quad (3.13)$$

where  $A_i$  is the amplitude of  $i$ -th particle,  $r_i$  is the wave particle radius,  $\mathbf{x}$  represents a point of the water surface,  $\mathbf{x}_i(t)$  is a the position of the  $i$ -th wave particle in time  $t$  and  $\Pi'$  is a box function, which limits cosine function over a finite region in three dimensional domain.

The drawback of the generalized deviation function in equation 3.12 and 3.10 is that the blending function depends on the direction of the neighbouring wave particles. Waveform function  $W$  from equation 3.10 for the 1D wave can also be used as the blending function  $B$  with the purpose of making the individual wave particles more flexible and less dependent on each other. The particle length is now replaced by the radius of the 2D wave.

Radial deviation function would not be sufficient by its own because earlier we assumed that the width<sup>1</sup> of the wave particle forming a non-linear wave front is changing with the wave front propagation. But now the length and the width of the wave particle are represented by the same parameter (radius  $r_i$ ). Radial definition does not allow changing the particle width independently on the particle length so it has to be constant in time. That means that we need to cover the whole size of propagating wave front with particles by placing one particle next to each other so that even with a constant particle size the wave front is continuous. Subdivision method enforcing this criterion will be derived in section 3.3.6.

Note that the radial deviation function is only an approximation to the generalized form discussed above. The approximation error is derived from the coverage of continuous

<sup>1</sup>Wave particle width represents the area of effect of the blending function described in equation 3.12.

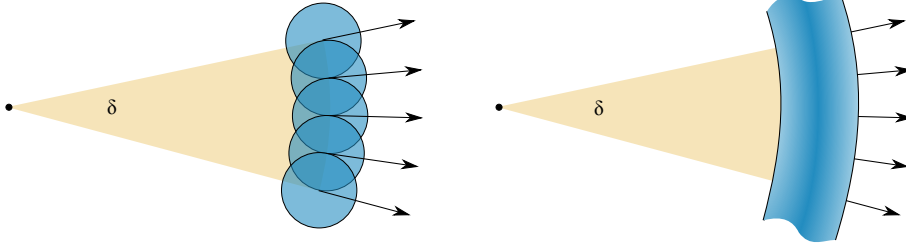


Figure 3.6: Illustration of the wave front generation from particles. Source particles (left) and generated wave front (right) from the top view.

expanding wave fronts. Note that the approximation error is bounded, which will be discussed in section 3.3.6. Moreover the radial definition allows using wave particles more efficiently.

### 3.3.4 Longitudinal waves

The motion of the water surface is not limited only to the vertical deviation. In reality water particles propagate in circles which creates sharper peaks on the surface waves as discussed in 2.3.1.

We show a method similar to the Gerstner wave model to the wave particle method to include the influence of longitudinal waves.

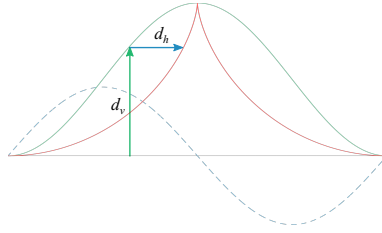


Figure 3.7: Illustration of the horizontal and vertical deviation function of a propagating wave.

$$D_h(\mathbf{x}, t) = \mathbf{x}_0 + \sum_{i \in \mathcal{P}} d_{h_i}(\mathbf{x}, t), \quad (3.14)$$

where  $D_h(\mathbf{x}, t)$  is the global deviation function,  $\mathbf{x}_0$  is the initial horizontal position of the surface.

The global horizontal deviation function represents the horizontal position of the surface after longitudinal wave takes effect.

$$d_{h_i}(\mathbf{x}, t) = d_{v_i}(\mathbf{x}, t) \mathbf{L}_i(\mathbf{u}), \quad (3.15)$$

where  $\mathbf{L}$  is a vector function which describes the longitudinal component of the wave. Vertical deviation function  $d_{v_i}(\mathbf{x}, t)$  is used in order to correspond to the shape of the original wave.

$$L_i(\mathbf{u}) = -\mathbf{v}_i \sin\left(\frac{\pi \mathbf{u}}{r_i}\right) \Pi'\left(\frac{u}{2r_i}\right), \quad (3.16)$$

where  $\mathbf{v}_i$  is the propagation direction and  $\Pi'$  is the box function.

Note that similarly to Gerstner waves amplitude must be chosen properly to handle the cases of self intersection. Therefore, the strength of the horizontal deviation  $s_v$  is introduced as global parameter. Similarly to the case discusses in section 2.2.1.1 intersections may occur if  $s_v k A > 1$ .

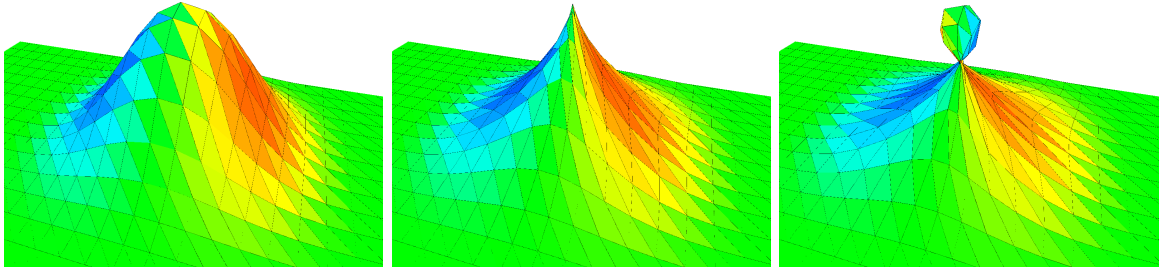


Figure 3.8: Illustration of the influence of the horizontal deviation on a existing wave particle. Wave produced by a single wave particle without the effect of the horizontal deviation (left). Other images (centre, right) show a case of different vertical deviation strength  $s_v$ .

### 3.3.5 Wave particle properties

In this subsection we describe the properties of individual wave particles with respect to the radial deviation function.

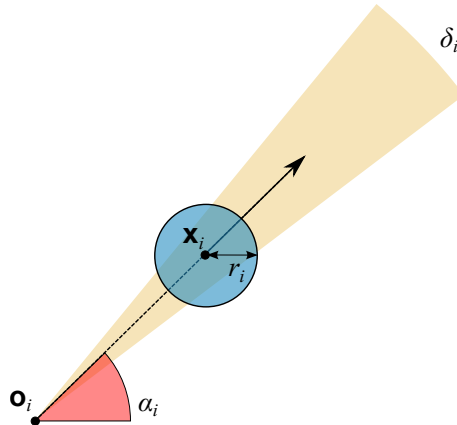


Figure 3.9: Properties of the  $i$ -th wave particle (blue circle) demonstrated from top 2D view. Yellow sector represents the dispersion angle  $\delta$ , red sector is the propagation angle  $\alpha$ ,  $\mathbf{x}$  is the current position,  $\mathbf{o}$  is the origin and  $r_i$  is the wave particle radius.

**Propagation angle** Wave particles propagate in 2D plane created by the water surface. Propagation angle  $\alpha$  represent the direction in which the particle moves. Linear wave fronts consist of particles which share the same  $\alpha$  in order to distribute consistently. On the other hand expanding or contracting wave fronts have different  $\alpha$  for each wave particle.

**Dispersion angle** Dispersion angle  $\delta$  is introduced to describe a spatial range in which new particles can be placed. It indirectly represents the curvature  $\kappa$  of the wave front in a place defined by the particle position.

$$\kappa = \frac{1}{r_\delta} = \frac{\delta}{w}, \quad (3.17)$$

where  $r_\delta = |\mathbf{x} - \mathbf{o}|$  is the radius of a notional circle describing the curvature, and  $w$  is the distance between adjacent particles as shown in figure 3.13. The use of  $w$  is discussed later in section 3.3.6.

**Origin** The wave particle origin is the position of the particle at time  $t = 0$  and it is fixed as the wave particle propagates. The dispersion angle forms a circular sector according to the wave front curvature with the centre in the wave particle origin. The wave particle is always located on an imaginary line going from the origin in the direction of propagation as show in figure 3.9. The particle origin is located behind the particle if it is a part of an expanding wave front. The opposite applies for contracting wave fronts and the origin can be seen as a focal point as demonstrated in figure 3.10.

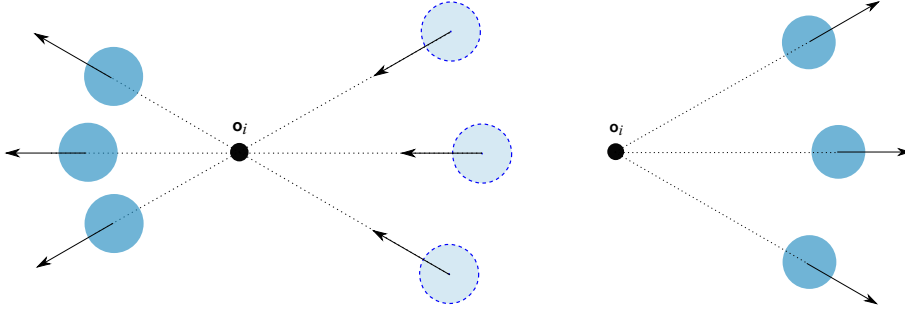


Figure 3.10: Illustration of the wave particle (dark blue circles) origin in the case of contracting (left) and expanding (right) wave front. The arrows represent the propagation angle,  $\mathbf{o}$  is the origin point, and the light blue dotted circles represent the wave particle in an initial position.

**Amplitude** Amplitude represent the energy of the wave particle. It contributes to the water deformation via the deviation function. The amplitude of a single wave particle is reduced according to the distance of wave propagation. Particles with low amplitude have also low contribution to the deviation function. To reduce the computational overload, redundant particles with amplitude lower than threshold  $T_A$  are deleted from the system.

In some scenarios it is useful to model waves with negative amplitudes. The application of negative waves is discussed in 3.4.2.2.



### 3.3.6 Wave particle subdivision

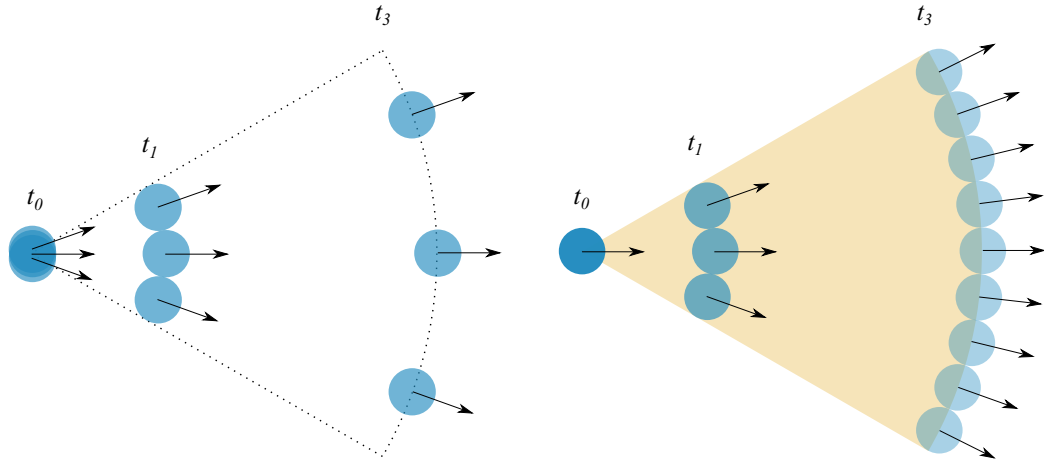


Figure 3.11: Illustration of the wave particle (blue circles) behaviour without subdivision (left) and with subdivision (right) in three time steps. The opacity of the wave particles colour illustrates amplitude. Yellow sector on the right image represents the dispersion angle  $\delta$  of the original (leftmost) particle. Note that the distances between particles should be even lower to form a continuous wave front; the particles are placed sparsely to increase readability. The difference in time  $t_0$  in both images is that there are all three particles placed in the same point on the left image. On the other hand there is only one particle which will be subdivided in the right image

As an expanding wave front propagates it covers larger area and since the particles have fixed size the distances between the particles increases. It will eventually increase to such a level that particles will be far from each other resulting in a non-continuous wave front. Empty spaces will arise in the positions where the particle density is low. Figure 3.11 shows an expanding wave front with radial wave particles propagating in time. Solution to this problem is a concept of a variable number of particles along the direction of wave front propagation.

The main idea in the particle subdivision method is to put two new wave particles to fill the hole in the wave front. If the distance between two neighbouring particles is larger than a defined threshold, a new particle is created on each side of the *parent* wave particle. The amplitude of the parent particle is distributed to the *child* particles in a way that the overall amplitude in the system remains the same.

Particle subdivision is used exclusively by expanding wave fronts because in a case of a contracting wave front the particles are moving closer to each other, creating a peak. After a contracting wave meets its origin, the dispersion angle turns over and the wave particle becomes part of an expanding wave front as shown in figure 3.10. In the rest of the section we will speak about expanding wave fronts.

An important property of the wave particle method's efficiency is that individual wave particles are independent and they do not know about the properties of other particles. In other words, we are not able to compute the distance between two arbitrary wave particles.

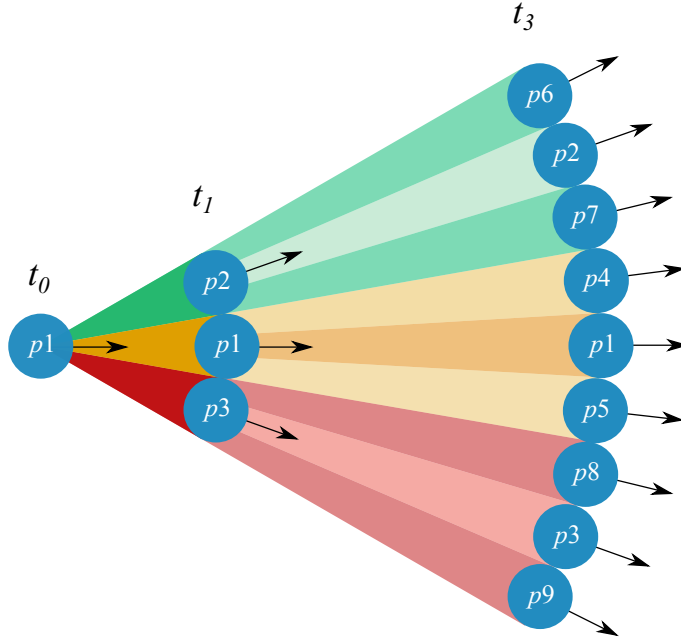


Figure 3.12: Illustration of the dispersion angle partitioning after particle subdivision operation. Particles are marked by blue circles with a unique identifier. Dispersion angle of each particle is marked by colour to enhance lucidity.

When a wave front is created we know both dispersion and propagation angles of each particle in this wave front. Since the particle velocity is constant we can compute in advance at which time the distance between neighbouring particles will be beyond threshold. The threshold is set proportionally to the particle radius  $r_i$ . It also means that neighbouring particles will never be further from each other than the threshold parameter.

### 3.3.6.1 Subdivision criterion

Note that distance on the circumference is used rather than Euclidean distance to take the curvature of the expanding wave front into account. Distance between adjacent particles  $w$  can be computed as follows

$$w = \delta r_\delta, \quad (3.18)$$

where  $\delta$  is dispersion angle, and  $r_\delta = |\mathbf{x}_i - \mathbf{o}_i|$  of the  $i$ -th wave particle. This means that as the distance of the wave particle from its origin increases, the distance between adjacent wave particles also increases.

Furthermore we can rewrite equation 3.18 in order to compute the distance in the time  $t$

$$w_t = w_0 + \delta \mathbf{v} |t - t_0|, \quad (3.19)$$

where  $w_0$  is the distance between neighbouring particles in the current time  $t_0$ , and  $\mathbf{v}$  is the particle velocity.

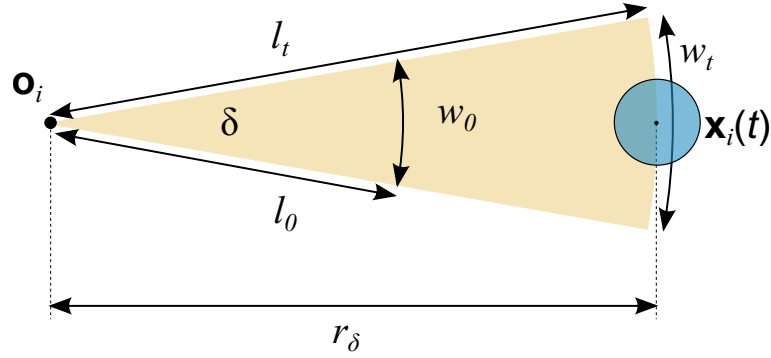


Figure 3.13: Visualization of the distances used by the particle subdivision method. The blue circle is the wave particle and the yellow region is the dispersion angle  $\delta$  in the time of particle creation when the distance travelled  $r_\delta = 0$ .

Threshold parameter  $T_w$  defines how often propagating particles subdivide. If

$$w > T_w$$

two additional wave child particles  $j, k$  are created on each side of the parent wave particle  $i$  as shown in figure 3.12.

The choice of the threshold parameter affects the shape of the propagating wave front. For lower values of  $T_w$  particles are created more frequently resulting into high particle density on the wave front curve. For a plausible results we choose  $T = r_i$  just as figure 3.6 illustrates. The threshold parameter also modifies the distance of the wave front propagation. Lower  $T_w$  leads to more levels of subdivision in shorter distance. This means that there will sooner be more particles with low amplitudes. After the amplitudes reach bellow  $T_A$ , particles are deleted resulting into shorter wave propagation.

Approximation error of the radial definition of wave particle introduced in section 3.3.3 is also affected by  $T_w$ . In fact the approximation error changes with the change of  $w$  and it results from the fact that the curvature of the wave front changes each step but the subdivision occurs in discrete step with lower frequency. Two neighbouring particles form a perfectly continuous wave crest if  $w = r_i$ . On the other hand a valley (peak) is formed if  $w > r_i$  (respectively  $w < r_i$ ). Both of these cases are shown in figure 3.14. The value of  $r_i$  comes from the radial wave particle definition. So the error of the wave shape can be bounded by the threshold  $T_w$  since  $w$  can not exceed it. Figure 3.15 shows the relation between above mentioned parameters. Note that  $w$  in this figure shows the distance between neighbouring particle although after each subdivision we refer to the newly creates neighbouring particles. In the very next moment after subdivision is performed the approximation error is zero and it satisfies the wave equation. Also note that the frequency of the subdivision gets lower as the particle propagates from its origin.

### 3.3.6.2 Creating new wave particles

**Amplitude** Important property of the convincing physical simulation is the energy conservation criterion. As mentioned earlier one wave particle can produce exactly two other

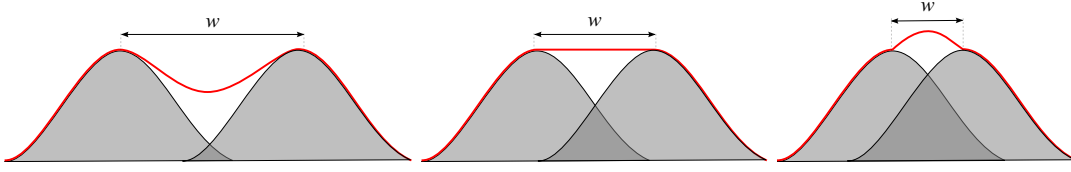


Figure 3.14: Exaggerated effect of the approximation error presented by the radial definition of the wave particle deviation function. Images are ordered by  $w$ . Wave front with no error (centre), wave front with undesirable valleys (left) and peaks (right).

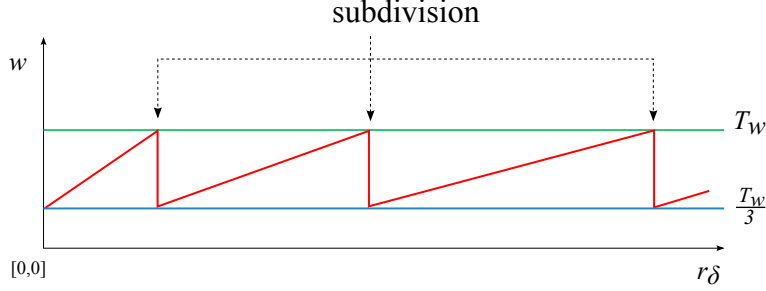


Figure 3.15: Illustration of the approximation error with respect to the distance between adjacent particles  $w$  and the distance travelled  $r_\delta$  by a particle.  $T_w$  refers to the subdivision threshold. The red line represents the approximation error. The green and blue line is the upper, respectively lower, bound of the error.

particles in one subdivision step. Moreover due to the fact that we defined the particle size to be constant it is necessary to delegate the energy of the wave particle to its children particles. In this case the energy is represented by the amplitude (deviation of the water surface).

The amplitude of the newly created particles  $k$  and  $j$  is

$$A_k = A_j = A_i = \frac{1}{3}\hat{A}_i, \quad (3.20)$$

where  $\hat{A}_i$  is the amplitude before subdivision, and  $A_i$  is the amplitude of the  $i$ -th particle after subdivision. We can see that even without any wave ambient dampening, amplitudes are slowly spread across the enlarging wave front which causes the wave front to slowly fade away.

**Dispersion angle** The same is valid for the dispersion angle since both the parent and the two children particles cover a non-intersecting sector of the parents dispersion angle before subdivision.

$$\delta_k = \delta_j = \delta_i = \frac{1}{3}T_w \hat{\delta}_i, \quad (3.21)$$

where  $\hat{\delta}_i$  ( $\delta_i$ ) is the dispersion angle before (respectively after) subdivision of the  $i$ -th particle, and  $T_w$  is the dispersion threshold.

**Propagation angle** Propagation angle of child particles is set according to the propagation and dispersion angle of the parent particle. It is necessary to cover the whole sector defined by the dispersion angle uniformly. To do so the propagation angle is adapted as follows

$$\alpha_i = \hat{\alpha}_i, \quad (3.22)$$

$$\alpha_j = \hat{\alpha}_i + \frac{1}{3}T_w \hat{\delta}_i, \quad (3.23)$$

$$\alpha_k = \hat{\alpha}_i - \frac{1}{3}T_w \hat{\delta}_i. \quad (3.24)$$

The  $\frac{1}{3}T_w$  comes from the uniform distribution requirement. Adjacent particles cannot be further from each other than  $T_w$ . If both such particles (e.g.  $p1$  and  $p2$  in figure 3.12) create one additional particle in between ( $p4$  and  $p7$  respectively) it is necessary to divide the distance  $T_w$  into three parts in order to maintain the uniform distribution.

**Position** Besides the uniformity new particles are placed on the imaginary circle represented by the dispersion angle  $\delta$ .

$$\mathbf{x}_i = \hat{\mathbf{x}}_i, \quad (3.25)$$

$$\mathbf{x}_j = \hat{\mathbf{o}}_i + r_\delta \delta_{xy}, \quad (3.26)$$

$$\mathbf{x}_k = \hat{\mathbf{o}}_i - r_\delta \delta_{xy}, \quad (3.27)$$

where  $\delta_{xy} = (\cos(\hat{\delta}_i), \sin(\hat{\delta}_i))$  is the 2 dimensional direction vector made out of scalar angle  $\delta$ .

**Origin** Origin is used for computation of  $r_\delta$  which is the equal for all particles which share the same ancestor particle. Exception to this is a case of wave particle reflection discussed later in 3.3.7.

$$\mathbf{o}_k = \mathbf{o}_j = \mathbf{o}_i = \hat{\mathbf{o}}_i, \quad (3.28)$$

### 3.3.7 Water boundary

In open ocean scenes with only one source of localized waves there is no need for handling collisions of the propagating wave. On the other hand scenarios such as pools or rocky shores require a model for reflecting incoming wave fronts off the water boundary. The boundary represents the container which holds the simulated water.

If we assume a flat boundary the reflection is processed simply by mirroring the wave particle propagation direction.

As mentioned in section 3.3.6.1 the distance from the particle position to the origin  $r_\delta$  is important for the particle subdivision. Assume the case shown in figure 3.16. Since the

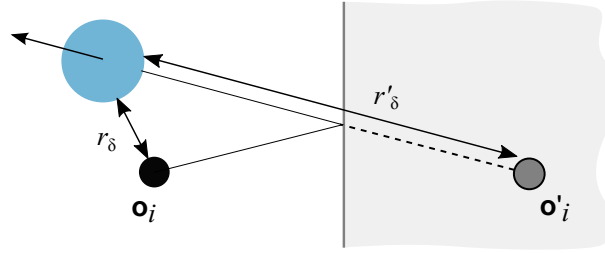


Figure 3.16: Illustration of the wave particle reflection. In this case  $r_\delta$  is wrongly interpreted as it does not cover the distance which has been travelled by the current particle.

particle has been reflected off of the boundary its propagation angle changed and all of a sudden  $r_\delta$  is decreasing instead of increasing with the travelled distance. This results into the particle is not subdividing as the reflected extracting wave front propagates. This can be solved by moving the origin of the wave particle inside the boundary in the same distance. In other word we also mirror the origin point along the axis defined by the boundary.

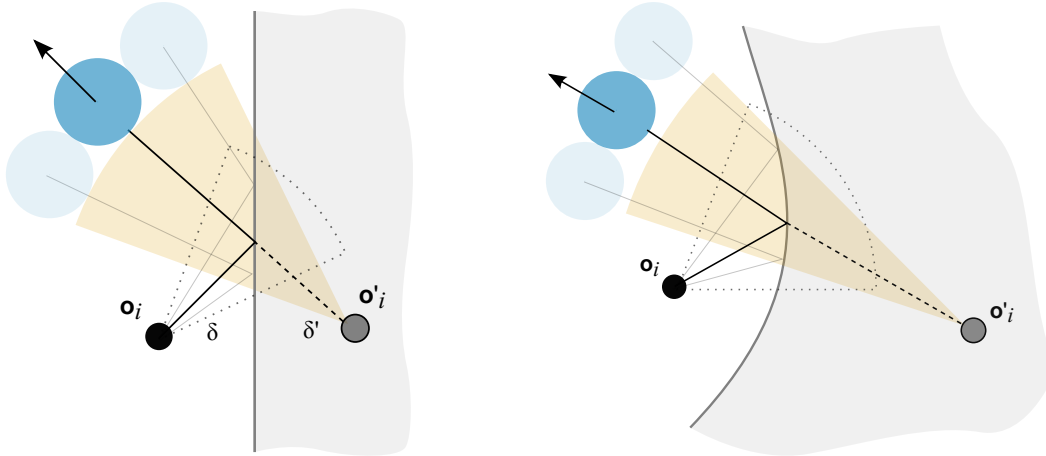


Figure 3.17: Illustration of the wave particle (blue circle) reflection off of the flat water boundary (left) and curved boundary (right). The grey area represents the boundary with the mirrored origin. The dotted sector represents the dispersion angle before reflection and the yellow sector is the dispersion angle after reflection.

Handling curved boundaries is more complicated case because the curvature of the boundary determines the change of particle distribution angle as shown in figure 3.17.

Since the distance between neighbouring particles  $w$  does not change during the reflection, the change of distribution angle can be express by relation

$$w' = w \quad (3.29)$$

$$\delta' = \delta \frac{r_\delta}{r'_\delta}, \quad (\text{wrt eq. 3.18}) \quad (3.30)$$

where  $\delta$  is the dispersion angle immediately before reflection and  $\delta'$  is the dispersion angle immediately after reflection; the same notation is used for other symbols.

Figure 3.17 depicts the change of dispersion angle according to the curvature of the boundary after reflection. Note that on the right image  $\delta \neq \delta'$  since the  $r_\delta \neq r'_\delta$  in equation 3.30. This is valid for both concave and convex boundaries.

**Wave front diffraction** When assuming arbitrarily shaped boundaries, wave diffraction effect may occur. In case of small slit in the boundary, wave front interacting with the boundary is split and the wave particles are separated. In order to handle this state the particle which crossed the slit has to change its dispersion angle. Nevertheless the problem is that we cannot detect such situation since the particles are independent thus the separated particle does not know the exact position of previously neighbouring particles.

Consider a wave front being separated into two parts. Without a model which supports wave diffraction phenomenon, solution formed by two disjoint parts of a wave front is not a valid solution to the wave equation. Therefore, in terms of the valid solution to the wave equation, we make an assumption about the boundary shape to evade the lack of the dispersion effect in the wave particle method.

We assume that inside the boundary there are no obstacles which could break the convexity of the boundary and separate the wave front. Corners of the boundary can also affect the wave front continuity. Consider a linear wave front approaching a corner containing obtuse angle as shown in figure 3.18. Part of the wave front reflect off of one edge of the corner and the other part off of the second edge and the wave front is again separated into two disjoint parts. This is true when the boundary contains corners with obtuse angle. Rectangular boundary is a special case where particle which reflect from one edge eventually reflect from the other one. This results into the wave front remaining continuous after the reflection.

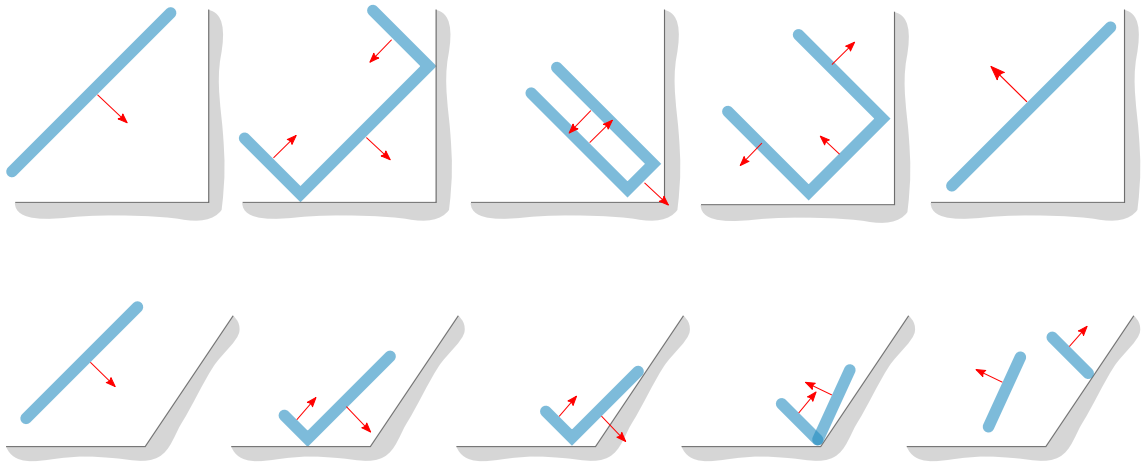


Figure 3.18: Demonstration of a wave front (blue line) approaching corner of water boundary in time (from left to right). Red arrows represent the propagation angle of the wave front. Valid reflection with the right angle corner (top) and reflection from an obtuse angled corner (bottom).

## 3.4 Object to fluid interaction

User interaction is a important part of the real-time simulation. Directly modifying water wave by the user interaction is not an interesting case because such interaction does not have a physical meaning. We aim a different interaction scenario where a rigid bodies are coupled with the wave generation process and the user can indirectly interact with the water using an object.

Modelling a physically correct water interaction simulation is a computationally demanding process since a volumetric approach is needed to take all the forces into account. Furthermore the rigid body can have a complicated shape on the fluid intersection. On the other hand physically correct model is not crucial in the real-time application rather than performance. We present some simplifications which allow us to efficiently detect placed where new waves emerge.

Static objects are not part of this sections because they can be handled as water boundary as discussed in 3.3.7.

First simplification is to separate the process of water interaction into two parts: *object to fluid interaction* and *fluid to object interaction*. In real life these processes are concurrent and cannot be separated. This separation is beneficial in terms of simulation because we can focus on the parts independently.

This section describes the process of adding dynamic waves into the simulations according to the motion of a rigid body in a fluid. The behaviour of the floating object is described in the following section 3.5 which aims the fluid to object interaction.

In real life the motion of a rigid body in water volume forces the water particles to move. This movement convects the velocity inside the volume which forms new waves. In contrast to that the simulation aims waves formed directly by the rigid body motion in the water rather than indirectly simulating the water flow and waves which it forms. This leads to a major simplification that waves are created immediately after the object hits the water. This means that waves are continuously created as the object moves in the water volume. Using this assumption turbulations in the water volume are completely ignored. The visual error produced by this simplification is large only for adjacent waves and the further the wave gets, the more the error decreases.

### 3.4.1 Wave-object collision

Floating objects also modify the behaviour of the wave propagating under the object. When a propagating wave encounters a floating object it enters a different medium and one part of the energy reflect and other one continues propagating. The change of behaviour depends on the wavelength of the propagating wave with respect to the size of the floating object. Waves with low wavelengths are easily reflected from the object and they have a small influence on the position of the floating object. On the contrary waves with high wavelengths move are almost not affected by the object. Note that this is also a simplification of the real-life phenomenon.

Due to the high number of wave particles in the system it would be computationally expensive to handle the collision of each wave particle with each floating object.



Therefore, we use a method based on superposition principle to cancel the effect of a wave under the object. We locate a particle under the floating object and add new *reducing particle* at the same location as the original one. Reducing particle has negative amplitude in order to locally decrease the global deviation function using superposition principle. The advantage of this method is that we do not need to address single particle nor compute intersection with the object.

### 3.4.2 Wave generation

This subsection describes the way of particle generation due to object interaction. Wave generation process is separated into three steps:

- Wave position,
- wave propagation,
- and wave volume.

#### 3.4.2.1 Wave position

We assume arbitrary object located in the water volume. When the object moves from a position to another it pushes certain amount of water in the motion direction. At the same time certain amount of water is pulled in the location where part of the object mass used to take place. This state is illustrated in figure 3.19. Consider the object being totally submerged into the water. Volume conservation principle ensures that the  $V_{pushed} = V_{pulled}$ . Even if the object is inside the water it can cause waves at the surface due to the convective flow of the fluid. The influence of the convective flow is suppressed exponentially by the distance from the water surface.

We distinguish two cases of the object's influence on the water surface: direct and indirect.

The upper part of the submerged object displayed in figure 3.19 has a direct influence on the water surface. That means there is no obstacle between the object and the surface. When a portion of water is pushed or pulled it will directly affect the surface.

The lower part of the object (figure 3.19) has an indirect contribution to the surface deviation, meaning the whole mass of the object prevents directly influence the surface. A portion of water induced on the lower part of the object has to be transmitted in order to influence the surface. The object boundary is a ideal place for the induced wave effect to take place because its the nearest location with a direct connection to the surface.

These two cases can be summarized such that the wave effect will influence the water surface from the place where it has been induced if it is on the top of the object. If not, the wave effect will be distributed to the boundary.

Silhouette of the object seen from a top view represents the object boundary in 3D domain. Another issue is to distribute the indirect wave effect to the object silhouette. Since we avoid using full 3D simulation we have no mechanism for determining the flow under the object. Therefore, indirect wave effect of each face is distributed uniformly according to the distance from the silhouette.

### 3.4.2.2 Wave propagation

In order to create wave front from the distributed indirect wave effect, the silhouette normal has to be computed. The normal vector of the 2D silhouette is used as the propagation direction of the wave particle in a point of the silhouette.

Since the simulation lacks the description of the flows in the water volume obtained by a full 3D simulation, we have no framework how to describe wave propagation based on the knowledge of its source. Therefore, we will establish basic rules for the propagation of a wave formed by an object motion in the fluid. These rules will be based on observation of real-life phenomenon. Results of the observation is generalized for the use in our simulation.

There are four distinct cases of the wave configuration after it is being created by an object motion. Figure 3.19 shows these cases in the context of the motion induced by the object motion. Cases where the object moves horizontally is a combination of the presented cases with vertical motion.

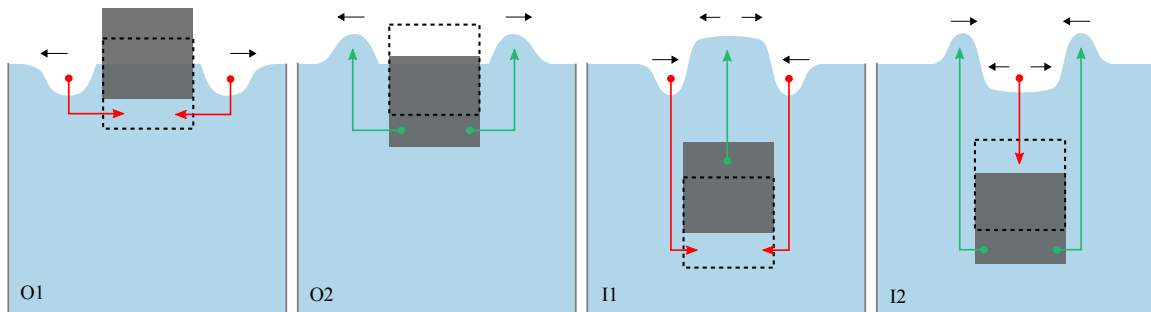


Figure 3.19: Different cases of wave propagation with respect to the position and the motion of the floating object. Striped line represents the object position in the previous time step. Cases O1, O2 (outside) show the influence of only the indirect wave effect since the object is on the water surface level. Cases I1 and I2 (inside) show the influence of both direct and indirect wave effect inside the volume.

The consequence of figure 3.19 can be summarized into two conditions: direction and orientation condition.

**Direction condition** Direction condition is specification of the wave propagation direction from the knowledge of the object motion.

- If the object is on the surface wave effect forms waves propagating in all directions away from the object.
- If the object is fully submerged into the volume it forms waves which move toward the object.

**Orientation condition** Orientation condition determines whether the wave amplitude is positive or negative in account of the object motion in the water.

- Amplitude sign of the indirect wave depends on whether it sinks (positive) or rises (negative).

- Amplitude sign of the direct wave depends on whether it sinks (negative) or rises (positive).

### 3.4.2.3 Wave volume

In section 3.4.2.1 it has been described how to locate the position of new particles as a part of the wave effect induced on the object surface. Each face of the surface contributes to the total volume induced by the object. Subsequently the total volume is evenly distributed across the object boundary.

Another problematic part is to determine the ratio between the amplitude and the radius<sup>2</sup>  $r_i$  and divide the volume appropriately. Since we do not have any other constraints we assume fixed size of the particle leaving the amplitude to be the dependent variable. This means that we cannot model waves with different frequencies natively by wave particle method. On the other hand superposition principle can be used to composite the result into lower frequencies.

### 3.4.3 Wave particle generation

This sections describes the actual process of wave particle generation. We will refer to the findings in the previous sections.

The process starts with the object motion in the fluid. Note that even static objects may induce waves when in the influence of the water flow. Therefore, we refer to the motion of the object relatively to the motion of the fluid. The water velocity at surface level can be captured easily using directly the wave particle velocity.

Each face of the object mesh displaces a portion of the water volume. The velocity direction determines whether the face pushes or pulls the water volume. If the water is pushed, positive wave effect volume is induced. Similarly if the face pulled the water volume, the effect is negative. The wave effect sign refers to the sign of the amplitude of the wave created in such way. The wave effect induced by the face  $f$  can be expressed as

$$V_f = A_f(\mathbf{v}_f \cdot \mathbf{n}_f)\Delta t, \quad (3.31)$$

where  $A_f$  is the area of the face,  $\mathbf{v}_f$  is the relative velocity,  $\mathbf{n}$  is the normal vector of the face  $f$ , and  $\Delta t$  is the time from the last step and is presented in order to include the length of trajectory from the last frame.

Wave effect induced by the faces which are deeper in the volume is scaled down exponentially. After this step the total wave effect induced by all the faces is known. The indirect effect is then distributed to the boundaries and smoothed out. Direction of wave propagation for both direct and indirect wave effect is covered in the previous section.

For each spatial portion of the wave effect a particle is created. In terms of using wave particles it is required to convert the wave effect into the amplitude. Volume of wave particle deviation is derived as a integral of the vertical deviation function defined in equation 3.13

---

<sup>2</sup>Wave particle radius refers to the wavelength of the original sine function used to represent the wave particle size.

$$V = \int_0^{2\pi} \int_0^r \frac{A}{2} \left( \cos\left(\frac{2\pi u}{r}\right) + 1 \right) u \, du \, d\theta = \frac{\pi}{2} A r^2, \quad (3.32)$$

where  $u$  is the distance  $|\mathbf{x}_i - \mathbf{x}|$ ,  $r$  is the wave particle radius,  $\theta$  is the revolution angle, and  $A$  is the amplitude. The multiple of  $u$  in equation 3.32 comes from the integration of cylindric revolution [Wei] of a function in the equation 3.13.

In the next step propagation direction and the dispersion angle are computed. In case of direct wave effect the wave propagates into all direction on the 2D surface. Particle which covers all the directions ( $\delta = 2\pi$ ) will subdivide into new particle in the very next moment. Therefore, the propagation direction can be chosen arbitrarily.

After the indirect wave effects distribution to the object boundaries, particles representing the indirect wave effect are placed on the object silhouette boundary as mentioned in previous section. Subsequently the propagation direction must be computed with respect to the silhouette curvature.

The dispersion angle  $\delta$  is also dependent on the silhouette curvature. With an increasing curvature, the sector which will be covered by the particle at the current position is enlarging.

Consecutively, wave particles can be finally put into the simulation. After that, wave particles lose their connection to their source and each other.

## 3.5 Fluid to object interaction

Section 3.4 described the process of generating new waves from the motion of the an object in a water. This sections depicts the influence of the water body to the object, which is submerged in it. The fluid influences the object in form of multiple forces.

### 3.5.1 Buoyancy force

Buoyancy is a force which is created by the fluid and opposes the gravitational force. Buoyancy force is created by the object submerged in water which due to the gravitational force pushes the molecules around it down in the water volume. Because of the low compressibility of liquids, pressure inside the volume rises resulting into a force which keeps objects with lower density on the water surface. Buoyancy force is described by the Archimedes law.

$$\mathbf{F}_b = -\mathbf{g}V\rho_f, \quad (3.33)$$

where  $F_b$  is the buoyancy force,  $\mathbf{g}$  is the gravitation acceleration,  $V$  is the volume of the submerged part of the object and  $\rho_f$  is the density of the fluid.

To compute the buoyancy force we need to know at least two variables from  $m = V\rho$  of the submerged part of the object. For simplicity we assume that objects are created from one material. This means that the object mass is uniformly distributed across the object volume. Using this assumption is it sufficient to compute the volume of the submerged part to obtain the submerged mass and vice versa. We chose to directly compute the submerged volume. We describe an approximative method of volume computation for the use in real-time graphics in section 5.4.1.

### 3.5.2 Dynamic forces

Dynamic forces originate from the relative motion of the object in the fluid. We distinguish two kinds of dynamic force: drag and lift force.

**Drag force** Drag force is exerted by the fluid friction and acts in the direction opposite to the rigid body motion. Drag force is based on similar principle as buoyancy force. With increasing velocity the object forces water particle to move resulting into higher pressure and slowing the object down. Drag force  $\mathbf{F}_d$  is expressed as [NAS10]

$$\mathbf{F}_d = \frac{1}{2} C_d A \rho_f (-\mathbf{v}^2), \quad (3.34)$$

where  $C_d$  is a drag coefficient,  $A$  is the area of the rigid body,  $\mathbf{v}$  is the velocity,  $\mathbf{v}^2 = v^2 \frac{\mathbf{v}}{v}$  and  $v = |\mathbf{v}|$ .

**Lift force** Lift force is the component of the dynamic force which cause the rigid body to move in the direction perpendicular to the motion. This means that the motion direction is changed based on dynamical properties of the rigid body. Lift force  $\mathbf{F}_l$  is formulated similarly to the drag force

$$\mathbf{F}_l = \frac{1}{2} C_l A \rho_f \mathbf{v}_\perp^2, \quad (3.35)$$

where  $C_l$  is the lift coefficient, and  $\mathbf{v}_\perp$  represent perpendicular vector to motion velocity.

Note that both of these equation 3.34 and 3.35 use a special coefficient. The shape of the rigid body determines these coefficients and there is no formula so they must be measured for each shape or simulated by a full 3D simulation.

We make an assumption that the total force applied on the object is the sum of the forces applied on each face of the object. This is rather radical simplification but it can be shown that it still offers a plausible solution which reflects the rigid body description.

Another simplification is to replace the parameters  $C_d$  and  $C_l$  with a constant which determines the intensity of the force. By this step we lose the dependence of the rigid body shape on the drag and lift force. To restore the dependence we use an effective area  $A_e$  parameter

$$A_e = A (\xi \mathbf{n} \cdot \mathbf{v}^u + (1 - \xi)), \quad (3.36)$$

where  $\mathbf{v}^u = \frac{\mathbf{v}}{v}$  is the unit velocity vector,  $\mathbf{n}$  is the face normal,  $A$  is the area of the face, and  $\xi$  is an user defined parameter which interpolates between the original face area and the area weighted by the direction of motion.

The last thing is to depict the lift force direction  $\mathbf{v}_\perp^u$ . As mentioned before lift force acts in the direction perpendicular to the propagation direction. In addition we assume that the lift force is also perpendicular to surface normal.

$$\mathbf{v}_\perp^u = \mathbf{v}^u \times \mathbf{n} \quad (3.37)$$



# Chapter 4

## Used technologies

The application is written in the C++ language using the OpenGL library. Algorithms used on the GPU are implemented in the GLSL shading language, which is part of the OpenGL library. In our application OpenGL is used both for simulation computing and rendering of results.

### 4.1 Other dependencies

In our application we use GLEW library with combination of GLFW to create OpenGL context. For assets handling we use the Assimp library to load 3D models and the Lodepng library to decode and encode png images. User interface is created by the AntTweakBar library. Parameters in the user interface can be also changed by an ini configuration file, which is parsed by a simple inih library.

Library	Version
Assimp	3.0.0
AntTweakBar	1.16
GLW	1.11.0
GLFW	3.0.4
GLM	0.9.3
inih	r29
devIL	1.7.8
noise1234	1.0

Table 4.1: List of dependencies of our application and their versions.

### 4.2 OpenGL library

OpenGL is a multi-platform API for hardware accelerated rendering of 2D and 3D graphics. Modern versions allow using a programmable pipeline on the GPU. The pipeline is divided into stages where the data are processed from the input to the resulting image. Shader is a programmable stage of the pipeline operating over specific data. Simplified OpenGL pipeline

is shown in figure 4.1. Some stages of the pipeline are optional so the figure shows that there are more paths to chose from.

In a typical scenario, vertex shader transforms the object vertices according to the virtual camera and passes them into primitive assembly. It can also modify other vertex attributes. If neither geometry nor tessellation shader is present, primitives are processed by the rasterization process. After rasterization, fragments are passed into fragment shader, where properties of the future pixels can be adjusted.

Geometry shader can be used to calculate per-primitive properties and even change the primitive type.

Tessellation shader is used to subdivide input primitives to create a finer geometric detail. The tessellation itself is implemented at the hardware level. The tessellation evaluation shader is used to set the tessellation level before the actual subdivision process. Tessellation control shader is used as control attributes of newly created vertices.

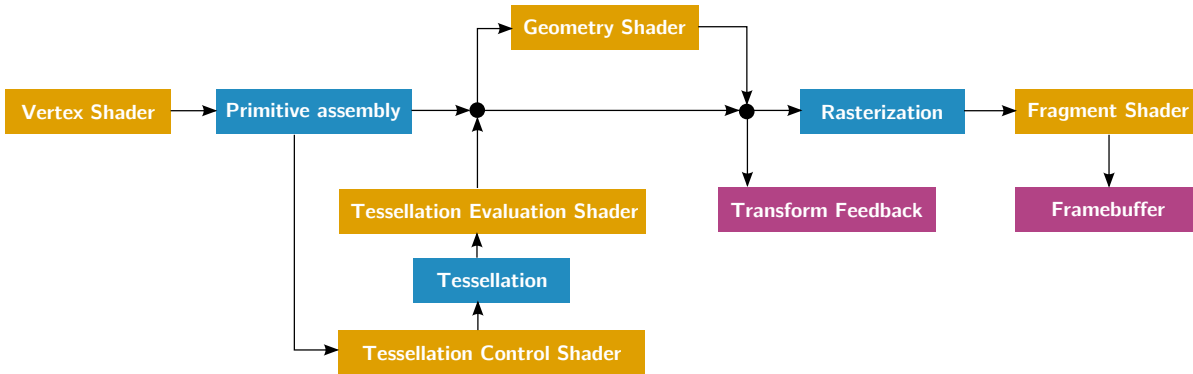


Figure 4.1: Simplified OpenGL pipeline. Yellow coloured boxes represents the programmable stages, blue are the fixed stages, and violet represents output buffers.

### 4.3 GLSL

GLSL is the programming language used in the OpenGL shaders. Although each shader stage operates with different data, GLSL presents an uniform interface for each of these shaders with some stage-specific functions.

The control flow of the program on the GPU is different from the control flow of the CPU architecture. In contrast of the CPU approach, branching may introduce performance overhead when part of the threads have a different flow than the other threads. In this case, both branches are synchronized and processed serially. Therefore, in our implementation we try to minimize the number of occurrences of non-uniform<sup>1</sup> branching.

<sup>1</sup>Branching with uniform variable, which is same for all threads.



# Chapter 5

## Implementation

In this chapter, we describe our implementation in detail. First we depict the structure of our application and used technologies. Following sections describe individual steps of water simulation using wave particle method on GPU. Rendering of the water surface is presented at the end of this chapter.

### 5.1 Application structure

Computer graphics application usually use object oriented approach to model the application entities similarly to the respective domain in real life. In our application we use a similar approach. However, large level of abstraction in the application may lead to performance drops. Therefore, we have designed simple model structure.

It is a common practice to design a `Drawable` base class to introduce a level of abstraction in the rendering loop. To delegate an input parameter to a uniform interface we introduce `GraphicsContext`, which is a structure that holds assets and global parameters that are required in the rendering stage of a drawable class.

Each drawable class in our application represents a single step in the whole simulation pipeline. We will describe each of these steps in detail in following sections.

- Wave particle simulation
- Filter particles
- Water to object interaction
- Object to water interaction
- Process rigid bodies
- Water optics
- Water surface

## 5.2 Wave particle method

One of the advantages of the wave particle method is its simplicity. It allows us to use an efficient implementation of such algorithm.

As mentioned before, wave particles do not require information about other particles and they are completely independent. Moreover, the interaction between water and rigid body is designed in a way that wave particles do not require information about the rigid bodies either. What makes our problem computationally complex is the large number of particles needed to represent a wave front. This means that we have a relatively simple problem, with a minimum of shared data but very large number of iterations. Since the GPU is designed for a high level of parallelism, this problem is more than suitable for a GPU implementation.

In terms of OpenGL, particles can be understood as points. Therefore, we decide to use OpenGL pipeline with programmable shaders to implement the simulation. This is suitable for our purposes because we can easily visualize these points and display the simulation results without using OpenGL Interoperability.

Considering OpenGL is mainly aimed for direct visualization, it is designed in a way GPU is reading lot of data, which are eventually displayed and thrown away. In our particle system it is essential to preserve the data on GPU memory without unnecessary data transfers from main memory to GPU buffer and vice versa.

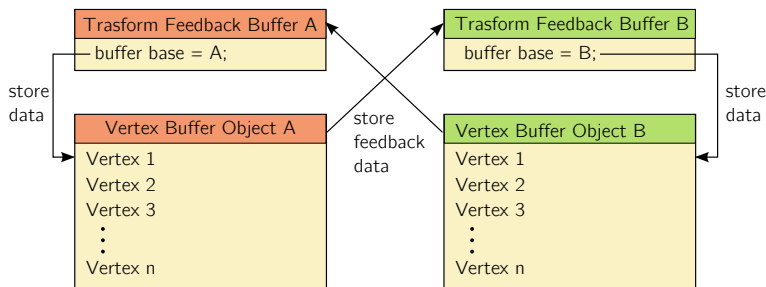


Figure 5.1: Illustration of Transform feedback buffer swapping. The output of first TFB is used as an input of the second TFB.

OpenGL Transform Feedback Buffer (TFB) allows us to capture the output of vertex or geometry shader inside the GPU memory. Location of TFB in the OpenGL pipeline is shown in figure 4.1. In each draw step the GPU fetches vertices<sup>1</sup> and pushes them in the vertex shader, where attribute properties can be modified or simply passed further in the pipeline. After that, the points can be stored in the TFB meaning that the data are persistent in one draw step.

Transform Feedback Buffer is rather a mapping to an existing buffer than a buffer by itself. The actual buffer where the primitives are stored can have different types. We use Vertex Buffer Object as the destination of Transform Feedback operation because we reuse captured vertices in the next frame.

Therefore, we use two Transform Feedback Buffers and we chain them together in a way that output of the first buffer is the input of the second buffer. The TFBs are connected

<sup>1</sup>Point is a 1D geometry primitive and can be represented by one vertex. Therefore, the term point and vertex is interchangeable in this context.

in the other way respectively. Two buffers are used due to the fact that OpenGL does not allow reading from the same Vertex Buffer Object (VBO) as is the target of the Transform feedback operation (read-write collision).

In each frame the vertices which represent our particles are sent into OpenGL pipeline. We use attribute data members to pack all the essential information to each individual wave particle. All the computations mentioned in section 3.3 are done on vertex shader. Only limitation in the vertex shader stage is that there is only one vertex in the input and one vertex on the output for one shader invocation. That means we cannot use vertex shaders for particle subdivision. Luckily geometry shader can handle this problem. Since the purpose of the geometry shader is different, it is less efficient in terms of performance. On the other hand geometry shader can emit new vertices, which are then also stored in transform feedback buffer.

Advantage of the Transform Feedback approach is that we do not address individual particles because the geometric topology is stored in TFB and the addressing is done at the hardware level.

Particle can be represented persistently on the GPU using a different approach. Particles can be stored in a preallocated array (texture), which contains particle information in each cell. While creating a new particle we need to find the array index to store its parameters. Since particles can be deleted, the array does not form a continuous structure which makes addressing even more difficult. Moreover, this process runs in parallel, so synchronization would be needed to handle array access collision.

### 5.2.1 Buffer size

As mentioned in section Transform Feedback captures processed geometry primitives into a buffer storage. The buffer size<sup>2</sup> is set prior to buffer allocation.

Transform Feedback does not have the information about the size of the output buffer without explicitly specifying it. Without this specification point generation is not limited by the buffer size resulting into undefined behaviour leading into overwriting the captured geometry.

Figure 5.1 shows the initialization of Transform Feedback Buffer in OpenGL. After specifying the buffer range, particles which would exceed the allocated size are not generated at all.

### 5.2.2 Particle data structure

In our implementation we represent parameters of the wave particles in attribute variables. Attributes variables represent additional data packed with the vertex. This means that the information is present in any part of the OpenGL pipeline and is stored in VBO along with position of each vertex. In order to avoid large memory consumption, we need to efficiently represent vertex attribute data.

---

<sup>2</sup>Buffer size reflects the maximum number of particles in the simulation.

---

```

1 GLuint vbo, tfb; // vertex buffer object, transform feedback buffer
2 glGenBuffers(1, &vbo);
3 glGenTransformFeedbacks(1, &tfb);
4 ... // set up vbo and attribute properties
5
6 glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, tfb);
7 glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 0, vbo);
8 glBindBufferRange(GL_TRANSFORM_FEEDBACK_BUFFER, 0, vbo, 0, nMaxParticles *
   sizeof(WaveParticle));

```

---

Source code 5.1: Initialization of the Transform Feedback Buffer.

Vertex attributes are internally packed and aligned into a multiple of `vec4`<sup>3</sup> in OpenGL[Ope14]. This means that it is beneficial to use `vec4` data type and fit all necessary information to it as few member variables as possible.

Figure 5.2 (left) shows the wave particle structure with all its parameters. The right figure shows our GPU packing of the wave particle structure.

- **Position** Since the wave particle propagates in a 2D plane aligned with the water surface, the position is sufficiently encoded into two floating point numbers each of them represent one axis of freedom.
- **Origin** In order to correctly compute the timing of the subdivision process the wave particle origin is required. Since we know the position and the propagation direction, we can reconstruct the origin from the knowledge of the travelled distance. Therefore, we use one float to store only the distance from the origin rather than a point in 2D plane.
- **Amplitude** The wave particle amplitude is simply stored in one floating point number.
- **Dispersion angle** To represent the angle of the wave particle dispersion, we use one floating point number.
- **Propagation direction** Direction in 2D domain can be also described by a single float representing angular coordinates.
- **Speed** Although section 3.2.2 describes an assumption of constant wave particle velocity, it is useful to add speed parameter to wave particle structure. The assumption is still valid for all wave particles representing one wave front. On the other hand, different wave front may have different propagation speed. Speed is encoded into one float.
- **Amplitude orientation** For implementation purposes we need additional boolean information to describe the sign of the amplitude. We need this information in order to represent wave dampening for both positive and negative amplitudes. Without this information a positive particle could become negative and vice versa. Since the speed is separated from the propagation direction, we need only the speed magnitude. Therefore, we use the sign of the speed to encode the amplitude sign.

---

<sup>3</sup>GLSL representation of 4-dimensional 32-bit floating point number.

- **Size** Wave particle size is the object of a similar assumption of constancy as the previous one. Adding this parameter allows us to create a broader wave front using lower number of particles. We encode the wave particle size as an half float into a floating point number. The other half represent half integer.
- **Number of consecutive border frames** Since this is a integer variable we can pack it along with the size as shown in ???. Detailed purpose of this field will be discussed in the section 5.2.4.

---

```

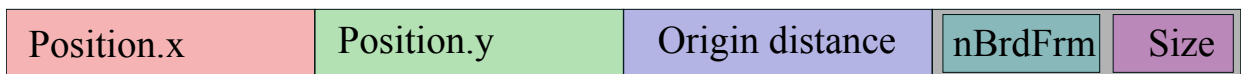
1 vec2 unpackSize(float compound){
2   float brdFrames = floor(compound / 65536);
3   float size = (compound - brdFrames * 65536);
4   return vec2(brdFrames, size);
5 }
6
7 float packSize(int brdFrames, float size){
8   return brdFrames * 65536 + size;
9 }

```

---

Source code 5.2: Data packing of the wave particle size

```
vec4 a_Pos2_Orig_Size
```



```
vec4 a_Ampl_Prop_Disp_Spee
```



Figure 5.2: Wave particle structure encoded for the use on the GPU.

### 5.2.3 Wave particle simulation

The implementation of the wave particle simulation routine is distributed into four parts according to the shader stage which handles it.

#### 5.2.3.1 Propagation routine

As mentioned previously in section 5.2 particles are treated as geometry consisting of points with attribute variables. Therefore, the entry point for the simulation process is in the vertex shader, where particles are properly placed and the parameters are passed further into the pipeline.

Figure 5.3 shows the vertex shader source code for the wave particle propagation. We choose using 2 `vec4` variables to represent a particle as mentioned in 5.2.2. We use bitwise operations in order to encode the `info` field shown in figure 5.2..

---

```

1 void particlePropagation(){
2   vec4 position = vec4(a_Position, 0, 1); // z coordinate is not important
3   float amplSign = sign(a_Speed);
4   float absSpeed = abs(a_Speed);
5
6   Out.f_Pos2_Orig_Size.zw = a_Pos2_Orig_Size.zw;
7
8   Out.f_Ampl_Prop_Disp_Spee.x =
9     a_Ampl_Prop_Disp_Spee.x - (absSpeed * u_Dampening * amplSign);
10  Out.f_Ampl_Prop_Disp_Spee.yzw = a_Ampl_Prop_Disp_Spee.yzw;
11
12  Out.v_Info = bvec4(false); // x = delete, y = create new
13
14  // Subdivision criterion
15  float w = a_DispersionAngle * a_OriginDistance;
16  Out.v_Info.y = w > T_w; // Current particle will be divided
17
18  // If the amplitude changes from + to - (delete it)
19  Out.v_Info.x = (Out.f_Amplitude * amplSign < 0);
20
21  ... // Boundary reflection
22
23  // Move wave particle
24  position.x += cos(Out.f_Ampl_Prop_Disp_Spee.y) * absSpeed;
25  position.y += sin(Out.f_Ampl_Prop_Disp_Spee.y) * absSpeed;
26
27  Out.f_Pos2_Orig_Size.z += absSpeed;
28  Out.f_Position = position.xy;
29 }
```

---

Source code 5.3: Main part of the wave particle propagation program on vertex shader

### 5.2.3.2 Subdivision and delete routine

Vertices are passed to the geometry shader, where the vertex can be either discarded or emitted. According to the input and output geometry specification, geometry shader may emit new vertices which are passed further into the pipeline. In other words the deletion process is done simply by not emitting the incoming vertex.

Deleting particles may occur from various reasons. The main reason is that the amplitude is lower than the threshold  $T_A$  as discussed in 3.3.5. The other two reasons are implementation dependent and are mentioned later in 5.2.3.3 and 5.2.4. The deletion detection is left to the vertex shader in order to increase the performance. The actual deletion has to be done on geometry shader so we keep the information in one bit of wave particle data structure.

Figure 5.5 shows the subdivision control on the geometry shader. The data structure of `tfOut` and `In` is shown in figure 5.2.

---

```

1 layout(points) in;
2 layout(points, max_vertices = 3) out; // Create max 2 additional vertices

```

---

Source code 5.4: Specification of input and output primitives in geometry shader.

---

```

3 ... // Declaration of input variables omitted
4 void main() {
5     if (doDelete(In[0].f_Ampl_Prop_Disp_Info)) return; // Delete particles
6
7     // Copy input to the transform feedback output
8     tfOut.f_Pos2_Orig_Spee = In[0].f_Pos2_Orig_Spee;
9     tfOut.f_Pos2_Orig_Spee = In[0].f_Ampl_Prop_Disp_Info;
10
11     bool doSubdivide = (feedOut.f_Info.g == CREATE_VERTEX);
12     float angle = In[i].f_DispersionAngle * ONE_THIRD;
13
14     if (doSubdivide) {
15         tfOut.f_DispersionAngle = angle;
16         tfOut.f_Amplitude = In[0].f_Amplitude * ONE_THIRD;
17     }
18     EmitVertex();
19     EndPrimitive();
20     // Create additional particles
21     if (doSubdivide) {
22         prepareNewParticle(+angle)
23         EmitVertex();
24         EndPrimitive();
25
26         prepareNewParticle(-angle);
27         EmitVertex();
28         EndPrimitive();
29     }
30 }

```

---

Source code 5.5: Geometry shader implementation of subdivision procedure.

### 5.2.3.3 Particle generation routine

Particle generation is the process of creating new particles based on the object to water interaction. The result of the interaction step is stored in the wave particle distribution texture. Wave particle distribution texture obtains information about spatial distribution of direct and indirect wave effect which is converted into particles in this step. Propagation direction is also part of the texture. The process of obtaining wave particle distribution texture is described in section 5.5.

Each pixel of the texture represents a potential wave particle. Therefore, we create a source vertex buffer and fill it with vertices organized into 2D grid with the same resolution as the texture. Consequently we set the wave particles properties to the vertices from the texture and we convert the wave effect to the amplitude as shown in equation 3.32. Delete flag of wave particles with zero amplitude is set. Once these particles reach the geometry shader they are discarded from the pipeline. On the contrary particles with non-zero amplitude and

---

```

3 void prepareNewParticle(float newDispAngle){
4   tfOut.f_Pos2_Orig_Spee.zw = In[0].f_Pos2_Orig_Spee.zw;
5   tfOut.f_Ampl_Prop_Disp_Info.x = In[0].f_Amplitude * ONE_THIRD;
6   tfOut.f_Ampl_Prop_Disp_Info.y = In[0].f_Ampl_Prop_Disp_Info.y +
       newDispAngle;
7   tfOut.f_Ampl_Prop_Disp_Info.z = abs(newDispAngle);
8   tfOut.f_Ampl_Prop_Disp_Info.w = In[0].f_Ampl_Prop_Disp_Info.w;
9
10  feedOut.f_Pos2_Orig2.xy = In[i].f_Position
11  // Subtract the parent propag direction from position => Origin
12  - (vec2(cos(In[0].f_PropagAngle), sin(In[0].f_PropagA)) *
13    In[0].f_Pos2_Orig_Spee.z)
14  // Add the new direction => Position
15  + (vec2(cos(tfOut.f_Ampl_Prop_Disp_Info.y), sin(feedOut.
       f_Ampl_Prop_Disp_Info.y)) *
16    In[0].f_Pos2_Orig_Spee.z);
17 }

```

---

Source code 5.6: Procedure of generating new particles on geometry shader.

a valid propagation direction are emitted and eventually captured by the transform feedback. This operation actually copies vertices from one buffer to another. Therefore, we can reuse the source vertex buffer in order to generate more particles.

Moreover, we organize the vertices in the vertex buffer in way that it can be reused while using differently sized textures. We assume that the distribution texture is a square texture with a power of two dimension. Figure 5.3 shows the data organization in the vertex buffer.

Figure 5.3: Illustration of the data organization in the particle generator vertex buffer.

### 5.2.4 Wave particle reflection

For performance purposes we represent boundaries as a texture. Normal vector of the boundary is encoded into each pixel of the texture in order to compute the reflection.

**Discrete boundary** Discrete step collision detection can produce errors when the object is moving too fast and the object passes through the boundary in one frame. We adjust the texture mapping with respect to the particle speed to handle these situations.

Another problem can arise when a particle subdivides. Imagine that the parent particle is moving along a boundary and subdivides. After the subdivision the child particle can appear behind the boundary as shown in figure 5.4.

We rather choose to handle the prone situation than the prevention. A prior collision detecting such as ray shooting is too computationally expensive for our use case. Instead we detect that the particle is behind the boundary and discard it. Data member `nBrdfFrames` from figure 5.2 represent the number of consecutive frames behind the boundary for each particle.



Figure 5.4: Illustration of a prone state of wave particle subdivision in the environment with water body boundary.

### 5.3 Filter particles

To perform a deviation of the water surface we need to render the wave particles. Particles are rendered as circles with the radius equal to particle size. Internally it is still a `GL_POINT` with a square shape but it is used with a texture sprite which defines the shape in the render.

The final deviation of the water surface can be obtained by rendering all the particles with additive blending from a top orthographic view similarly to the texture splatting. The radius directly influences the number of generated fragments. The performance of the blending operation depends on the number of fragments which result into single pixel. For larger particle radii this approach leads to a significant performance drop.

Instead we use a smaller points to represent the information about the particle presence and render them into texture. The wave particle render texture is filtered and the contribution of each wave particle in a local distance is accumulated. This is similar to the texture gathering process.

Figure 5.5 depicts the difference between these approaches. Note that the particle size and particle radius denote the same thing in the first case. On the other hand, in gathering approach the particle size represents the actual point size while the radius denotes the filtering distance.

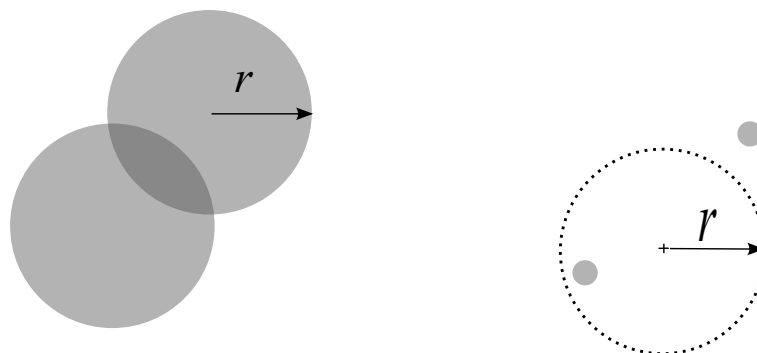


Figure 5.5: Illustration of splatting (left) and gathering (right) approach.

Nevertheless, even with a small particle size it is likely that the new particle overrides a previously rendered particle. Specially due to the nature of particle generation. Therefore, we use blending for point rendering to avoid the overriding.

In this step wave particle deviation function is applied. The contribution of each wave particle in the filtering step is weighted by the deviation function mentioned in 3.13. Advantage of the radial definition of wave particle deviation is that the filter kernel of such function is separable. This means we can perform 1D filtering process consecutively for each axis and compose the final result. Compared to the 2D convolution separable filters reduce the number of texture lookups to increase the performance.

The filtering process is implemented on the fragment shader, where the particle render texture is read in a local neighbourhood in  $X$  axis. Consequently the result is stored into `filtered_by_x` texture. The texture is used as the input of the second stage, where the filtering by  $Y$  axis occurs. The size of the filtering kernel is a global user defined parameter.

The filter function can be denoted as

$$d_h^X(p) = \frac{1}{2} \left( \cos \left( \frac{\pi p}{r} \right) + 1 \right), \quad (5.1)$$

$$d_h^Y(p) = \frac{1}{2} \left( \cos \left( \frac{\pi p}{r} \right) + 1 \right), \quad (5.2)$$

where  $d_h^X(x)$  is a  $X$ -axis horizontal deviation filter function,  $r$  represents the radius (kernel size), and  $p = [-r, r]$  is the distance of a pixel to the kernel centre. The same notation is valid for  $Y$ .

The case of horizontal deviation is more difficult because it depends on the wave particle propagation direction. As discussed in the original article we cannot simply blend the directions because the horizontal deviation is computed independently per each wave component of the final superposition. Therefore, we use an approximation where the propagation direction in equation 3.16 is replaced by the filter shift. In other words the horizontal deviation pushes the wave to centre of the wave particle as shown in figure 3.8. This results into the horizontal deviation is applied on each individual wave particle rather than on the wave front. The approximation slightly breaks the continuity of the wave front crest but can be compensated by subdivision error and the threshold  $T_w$  discussed in section 3.3.6.1.

$$d_{v_x}^X(p) = -\frac{1}{2} \sin \left( \frac{\pi p}{r} \right) \left( \cos \left( \frac{\pi p}{r} \right) + 1 \right), \quad (5.3)$$

$$d_{v_x}^Y(p) = \frac{1}{4} \left( \cos \left( \frac{\pi p}{r} \right) + 1 \right)^2, \quad (5.4)$$

$$d_{v_y}^X(p) = \frac{1}{4} \left( \cos \left( \frac{\pi p}{r} \right) + 1 \right)^2, \quad (5.5)$$

$$d_{v_y}^Y(p) = -\frac{1}{2} \sin \left( \frac{\pi p}{r} \right) \left( \cos \left( \frac{\pi p}{r} \right) + 1 \right), \quad (5.6)$$

where  $d_{v_x}^X(p)$  is a filter function for the  $x$  component of the vertical deviation in the  $X$ -axis.

Surface normals may also be computed at this step from a surface gradient. We can obtain the gradient by filtering the particles with kernel derived from the analytical derivation of the vertical deviation function. In our application we compute the normals on existing height field because we use more sources of the surface deformation.

## 5.4 Water to object interaction

All the forces acting on the object in the fluid in our simulation are computed by this module. We distinguish four types of forces, all of which have similar implementation details.

Throughout the application we use *silhouette camera*, which is a orthographic virtual camera capturing the rigid body from a top view. It follows the object while moving and the projection tightly the bounding box of the rigid body.

Each of the forces is computed on the GPU and the result is stored in a texture rendered from the silhouette camera.

### 5.4.1 Buoyancy force

For computing the buoyancy force we need to compute the part of the object which is submerged in the water. Deriving the force from the underwater volume fraction is described in section 3.5.1. In the initialization phase of our application we use the same technique to compute the volume of the whole object by moving it below the water surface plane.

We describe an implementation of an approximative method of 3D mesh volume computation. In the first step we render the rigid body by the silhouette camera using additive blending. In the fragment shader, fragments which are above the water level are discarded. Consequently each face of the object contributes to the overall volume accordingly to its orientation. As shown in figure 5.6, faces which are oriented down form the bottom part of the object while the faces oriented up form the upper boundary of the object. The distance from the water surface to the bottom boundary forms a positive volume while upper boundaries contribute with negative volume. After the rasterization process each fragment represents the depth of the object boundary along a single ray.

Figure 5.6: .

---

```

1 vec3 normal = normalize(In.normalObj);
2 int orientation = (int(normal.z < 0) * 2) - 1; // up = 1, down = -1
3 v_FragColor.x = orientation * (In.position.z - u_WaterPosClip.z);

```

---

Source code 5.7: Volume computation on the fragment shader.

Consequently the values from the texture are summed together and used to compute the buoyancy force on CPU. Note that the result is not in world coordinates. We convert the depth from the clip to the object space by using inverse projection matrix of silhouette camera. Moreover, pixel size is expressed in the world coordinates from the projection matrix.

### 5.4.2 Drag and lift force

Simplifications used in the drag and lift force computation are described in section 3.5.2. We use GPU implementation to compute forces acting on each face of the object mesh. Centroid of each face is computed and uploaded to GPU as a point. Each vertex computes drag and lift force with respect to the equation 3.34 and 3.35. The object velocity is passed into the shader as uniform variable. We use `glPointSize(1)` to ensure that each vertex will be rasterized into one fragment. Similarly to the buoyancy texture, fragments which are above the water surface are discarded since they do not affect the floating object. Forces are encoded as colours and rendered into a texture with additive blending.

### 5.4.3 Water-object collision

When a floating object encounters a wave front, part of the energy is transformed into a kinetic energy. Nevertheless the object motion is mainly induced by the relative motion of water beneath the floating object.

This force is calculated from the wave particle render texture. In addition of height value we also render wave particle propagation direction and speed in the remaining colour channels of the texture. Again we sum the texture in order to obtain the final force. While summing the forces we use buoyancy texture as an object silhouette stencil to precisely select which particles are affecting the object.

### 5.4.4 Parallel reduction

The common operation in the previously mentioned methods is the part where the texture is summed and the results are used on CPU. Although aforementioned texture does not require high resolution, transferring the whole data to the client memory in each frame would be time consuming. Therefore, we use a compute shader implementation of parallel reduction to sum the texture on the GPU and pass only the result to the CPU.

## 5.5 Object to water interaction

This section describes the implementation behind the object to water interaction. We have covered the process of the particle generated based on the wave particle distribution texture in section 5.2.3.3. This section denotes the distribution texture generation on the GPU.

In order to create wave particle distribution texture we precompute three additional textures, all of which have the same resolution. Additionally in each step of the algorithm we discard all the fragments that are outside water because they have no influence on generated waves.

**Silhouette texture** In the first step we use the silhouette camera to render the object seen from a orthographic top view to a *silhouette texture*. Since we will generate a particle for each pixel of the particle distribution texture, the resolution is set so that the pixel size in the world coordinates is similar to the size of the wave particle.

We discard all the fragments above the water level and render the depth in the water and orientation of the normal. We are interested only in the  $z$  component of the normal which tells us whether the pixels represents inner or outer part of the object. We also add a small bias to the normal in order to categorize faces which are perfectly perpendicular to the camera view. Moreover, we disable the face culling in order to see the inner part of the object.

**Wave effect texture** Consequently we upload face centroids on the GPU and query the silhouette texture. Figure 5.8 shows vertex shader snippet which recognizes faces with direct or indirect wave effect. Part of the vertex attribute data is the face area. Wave effect is rendered into the wave texture according to the equation 3.31. Note that blending is used.

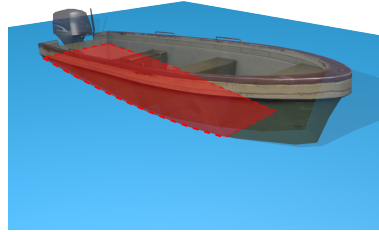


Figure 5.7: Red area represents the silhouette of the floating object.

---

```

1 // x = depth of the surface in water
2 // y = z-component of the normal
3 vec4 silhouette = texture(u_SilhouetteTex, Out.position.xy * 0.5f + 0.5f);
4
5 if(silhouette.y > 0){ // face is submerged
6   // Is the centroid depth larger than silhouette depth?
7   if((Out.position.z - u_WaterPosClip.z) - 0.01 > silhouette.x){
8     // Face is not on top => Indirect
9   } else {
10    // Face is on top => Direct
11  }
12 } else {
13   // Face is on the bottom => Indirect
14 }

```

---

Source code 5.8: Wave effect categorization of individual faces on vertex shader.

**Contour texture** In this step we recognize the contour of the object silhouette where particles with indirect wave effect are placed. Also the normal of the contour is included in order to assign propagation direction to the future particles.

The contour represents the boundary of the object part that is not submerged in water. Examples of object contour are shown in figure 5.8. A pixel represent a contour point if it satisfies the contour condition (shown in figure 5.9) and at least one adjacent pixels does not satisfy the condition. Pixels with neighbours that satisfy the condition form the inner part of the contour.

---

```

1 int isSilhouetteContour(vec4 silhPix){
2   // underwater && backface
3   return int(silhPix.x > 0 && silhPix.y <= 0);
4 }

```

---

Source code 5.9: Recognizing the silhouette contour pixel.

There are special cases when this method fails to recognize the contour. Since this method operates on image projected from an orthographic top view, it cannot handle models with valleys in vertical direction. Such models form an additional contour in locations where the water plane intersects the model as shown in figure 5.9.

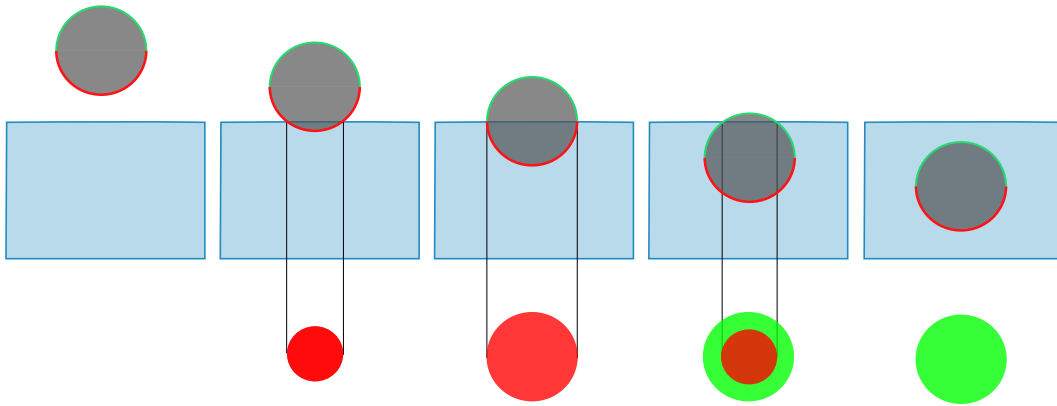


Figure 5.8: Upper image row represents a water tank seen from side view. We show different positions of a sphere relative to the water level. Note that the sphere is green on the top (positive  $z$ -coordinate of the normal) and red on the bottom (negative normal). The bottom row shows the same object seen from a top view without the part which is above the water level. The silhouette is the union of the green and red part while the contour is outer border of the red part.

In order to handle this limitation we assume using models which are monotonous in the vertical direction. In other words we use models which do not have holes in the vertical axis. Benefit of using such models is discussed in section 5.6.

**Distribution texture** The last step of the object to water interaction stage is distributing the indirect wave effect to the object contour. This routine also smooths out the contour normals in order to uniformly cover the circular area around the object by the wave particle propagation angles.

Similarly to a parallel reduction approach we sum up neighbouring pixels into one. In each step we merge four adjacent pixels into one pixel. After few iterations when the texels are summed together the process is reversed and we reconstruct the original silhouette while using the textures from the intermediate steps. While descending to higher texture resolutions we distribute the total indirect wave effect pixels recognized as contour pixels.

## 5.6 Floating object

Each object which is part of the water simulation interaction is aggregated into `FloatingObject` class. The class aggregates every instanced information which is used by the rest of the simulation. Floating object is actually composed out of the object. The first object is used for solid body rendering while the `convex` object is used for object to water interaction. This is beneficial because we can choose from a simpler model for higher performance or finer model for higher resolution. The `convex` is not necessarily convex. The important property is the aforementioned

Besides this, floating object aggregates `sillhouetteCamera`, `particleCamera` and affiliated textures. Floating objects even store the identifier of the vertex buffer object and

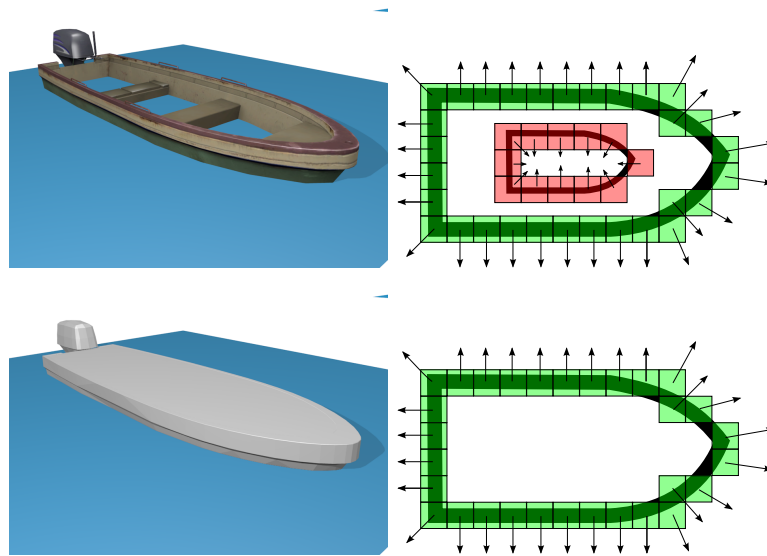


Figure 5.9: Illustration of the contour generation. Object (above) is not monotonous by z-axis a forms an prone contour. The object bellow forms a valid contour. The arrows represent 2D normals of to contour.

transform feedback buffer. This means we allocate a OpenGL buffer for each floating object.





# Chapter 6

## Results

In this section we will present results of our simulation. We depict different use cases of water simulation and show that the wave particle technique can be used to model such scenarios.

### 6.1 Simulation scenarios

We tackle three main simulation scenarios: Open oceans where nothing else than water is around, near shore area where interaction between other ships may occur, and finally water pools with limited boundaries. Render of simulation scenes from our application is seen in figure 6.1.

### 6.2 Quality testing

We have captured a simple testing scenario in real life a compared it to the simulated testing scenario.

### 6.3 Performance testing

We have tested our application on the following configuration.

- Intel Core i5-4590 CPU, 3.30GHz
- GPU: GIGABYTE GTX970 4GB
- RAM 8.0 GB
- Windows 7 64-bit
- Compiler: Visual C++ 18.00

We have used two timers to measure the performance, both of them are on the GPU. In order to measure frame rates we used asynchronous approach using two timer queries on the GPU. Code in figure 6.1 depicts the usage of such approach. This ensures that the measured GPU commands are finished by the time of the query and no synchronization is needed[[Lig](#)]. Therefore, is not slowed down by the time measuring. Except for the frame rate we measured the performance of each stage of our simulation. Since the previously mentioned solution measures the time of the whole frame we have used the synchronous timer query to measure each stage.

---

```

1 glBeginQuery(GL_TIME_ELAPSED, query[frontBuffer]); // queryBackBuffer
2 // Code to measure
3 glEndQuery(GL_TIME_ELAPSED);
4 glGetQueryObjectui64v(query[frontBuffer], GL_QUERY_RESULT, &timer);
5 // Swap buffers

```

---

Source code 6.1: OpenGL asynchronous time measuring.

We implemented a simple solution for reproducing the user inputs in each test case. Each floating object is given a set of impulses to either accelerate or turn together with a information about timing.

We have tested the performance of each simulation step in four testing cases.

**Test case A** The first case is a single boat floating in the ocean scene. We fix the length of the simulation to 2000 frames and count the time of each simulation phase. The ship is forced to move around the scene in a circular motion. We performed three modifications to this test case. In the first run we test the influence of rigid body complexity on the performance of each phase. For example computation of drag and lift force is done per each face of the rigid body. Figure 6.7 shows the testing results.

Similarly we measured the influence of the both silhouette (figure 6.7) and wave particle texture (6.8) on the application performance. Note that this parameter affects not only the texture resolution but also indirectly affects the number of fragment shader invocation etc.

**Test case B** In the second test case particles are added to the buffer until it is full. This test cast measures the performance of the wave particle propagation procedure. Note that in the wave particle method the number of particles is not directly proportional to the quality of visual result. Especially when most of the particles in the system are the ones with low energy.

Figure 6.8 shows the performance of the each algorithm stage with respect to the number of wave particles.

During the implementation, we measured the performance while using different wave particle data structure. We measured the usage of 64 byte<sup>1</sup> wave particles and compared it to the current 32 byte implementation described in 5.2.2. Unfortunately the performance tests did not show any signs of optimization.

---

<sup>1</sup>Four vec4 member variables, each of which is composed of 4 floats and float is a 4 byte data type.  $4 \times 4 \times 4 = 64$ .

**Test case C** We put the third test case into the lighthouse scene and place boats in the nearby. The purpose of this test is to show the usage of the wave particle method in a real simulation scenario. The rest of the boats in the environment does not accelerate by itself and is floating on a constant position. We position the boats in order to maximize the number of interaction between nearby floating object. Once a moves, it creates wave which pushes away the other floating objects. We measure the performance for different number of boat along the direction of motion.

**Test case D** The fourth test case captures a scenario with high number of boats. Unlike the test case C, all the boats are moving. Results are shown in figure 6.9. In our application, the number of ships is somewhat limited. Because we render the water surface in one step, all the vertical deviation functions are added at once. And since the number of GPU texture units is limited we cannot apply deviation function from more ships than `GL_MAX_TEXTURE_IMAGE_UNITS`.




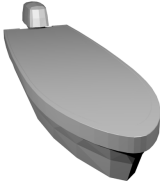
	Sphere x20	Sphere x60	Boat x20	Boat x60
				
Used method	32912,570313	888638,812500	4254,864746	114881,203125
Precise	32795,530	890339,293	3745,223	101121,294

Table 6.1: Evaluation of the approximative volume computation method used in our simulation.

## 6.4 Volume testing

Since the volume computation method is based object projection it is discretized into pixels. It means that the method is not consistent for different scales. Therefore, we present different scales of the object in the volume testing. Table 6.3 show results of our testing.

Ship count	1	2	4	8	16	32
Test case D	199	144	93	60	35	18
Test case C	145	136	97	60	33	24

Table 6.2: Average frame rate for the test case C and D.



Figure 6.1: Simulation environment rendered in our application

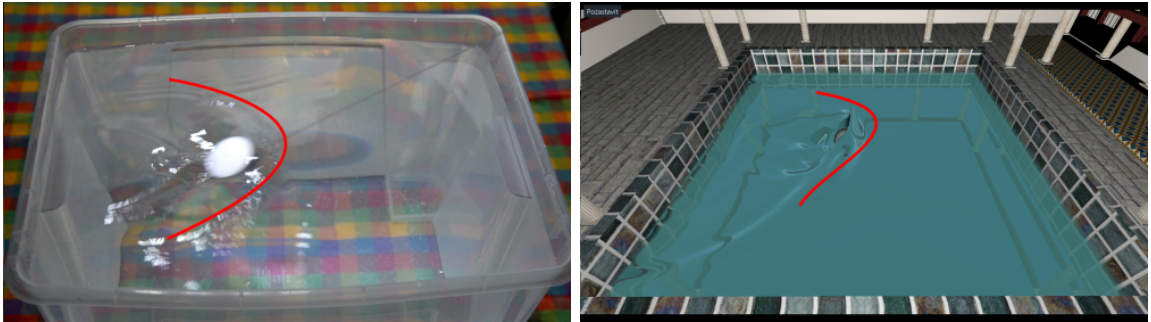


Figure 6.2: Comparison of wave propagation of real life scene with a simulated scenario.



Figure 6.3: Video sequence of real wave propagation in a water tank.

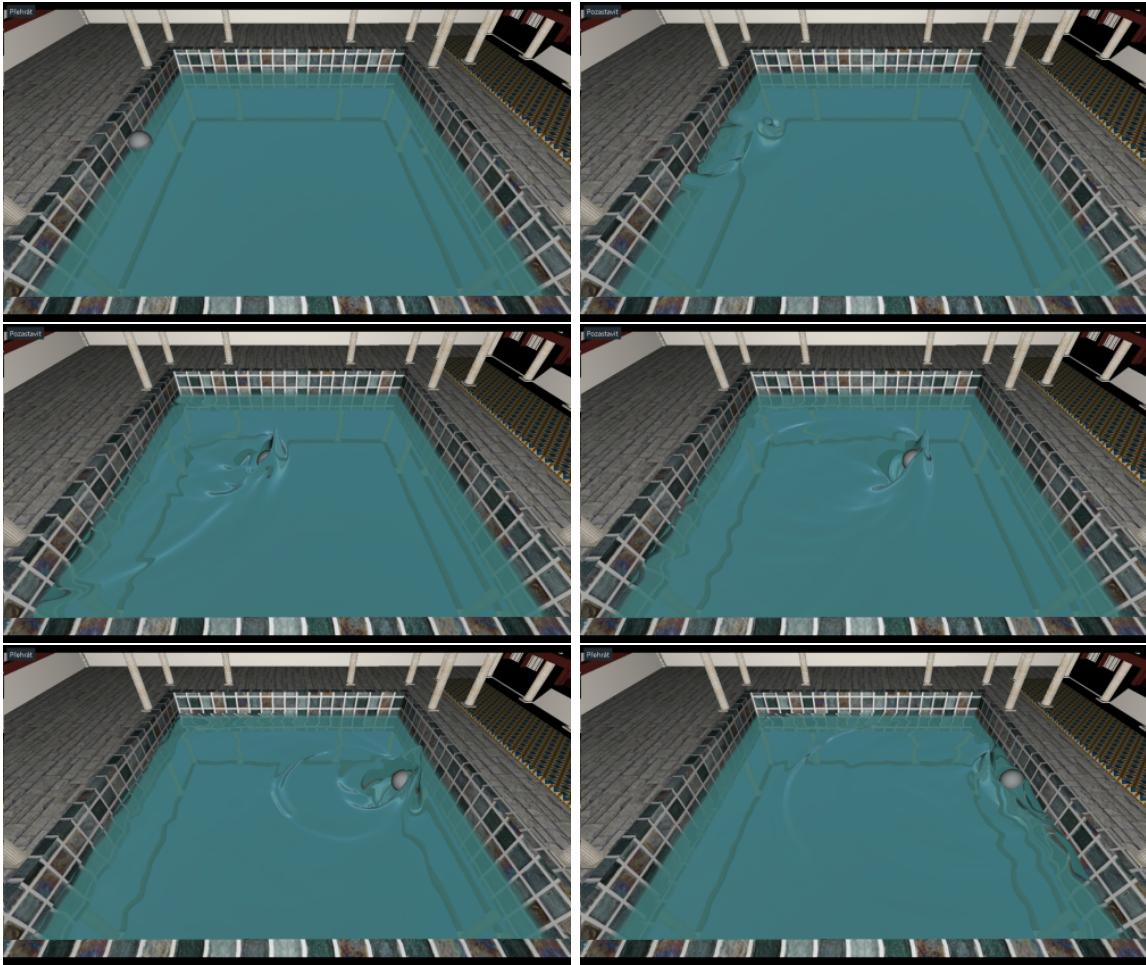


Figure 6.4: Video sequence of simulated wave propagation in a pool.

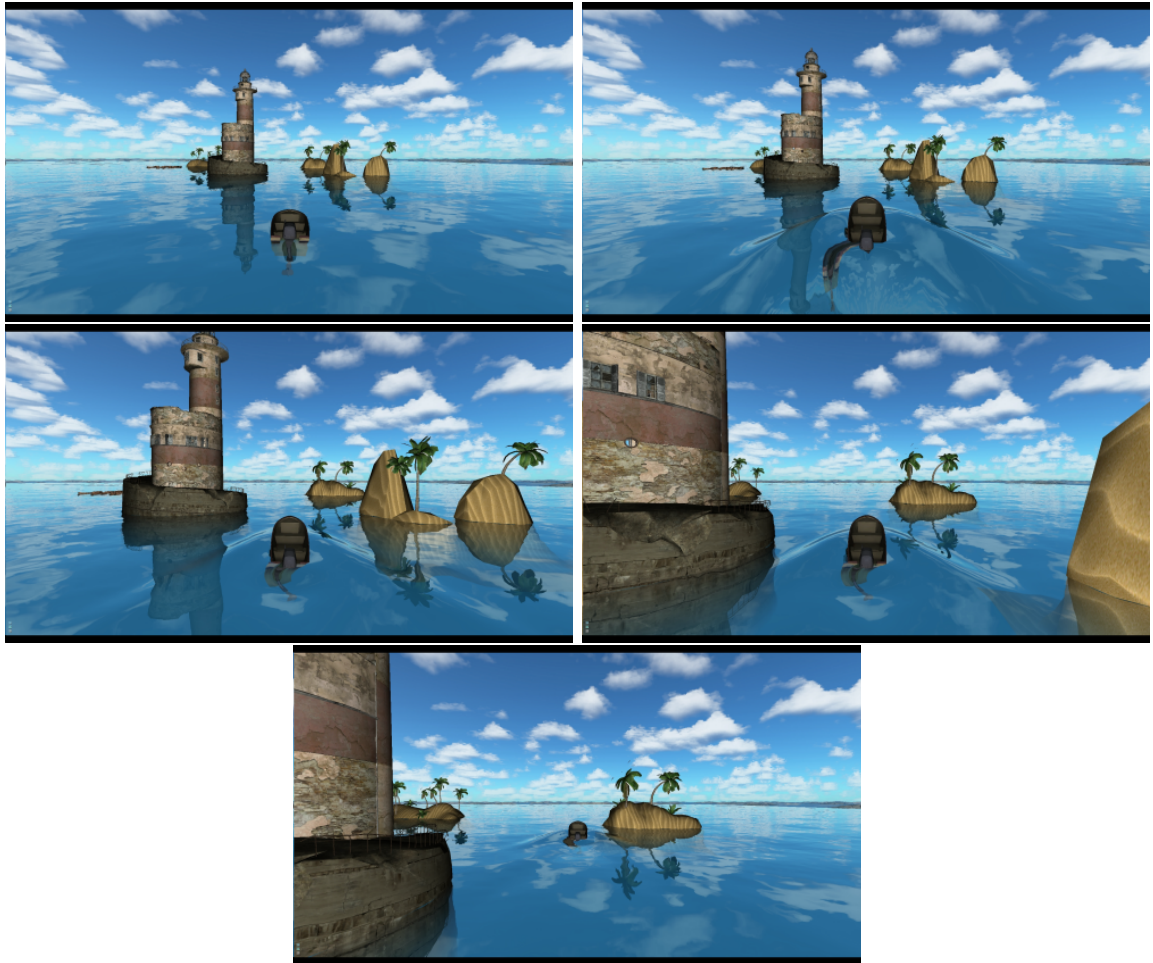


Figure 6.5: Video sequence of the floating boat in lighthouse scene.

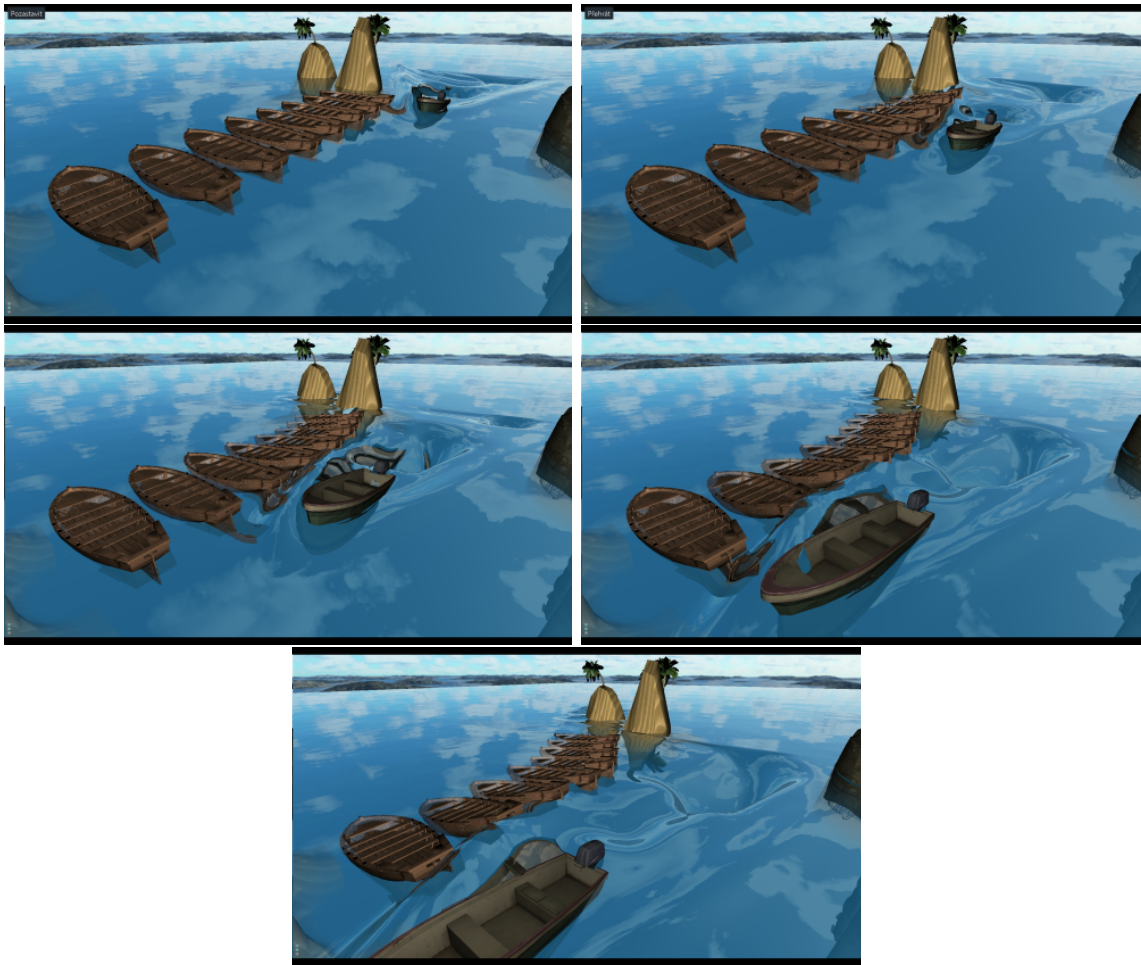


Figure 6.6: Video sequence of the boat interaction in lighthouse scene.

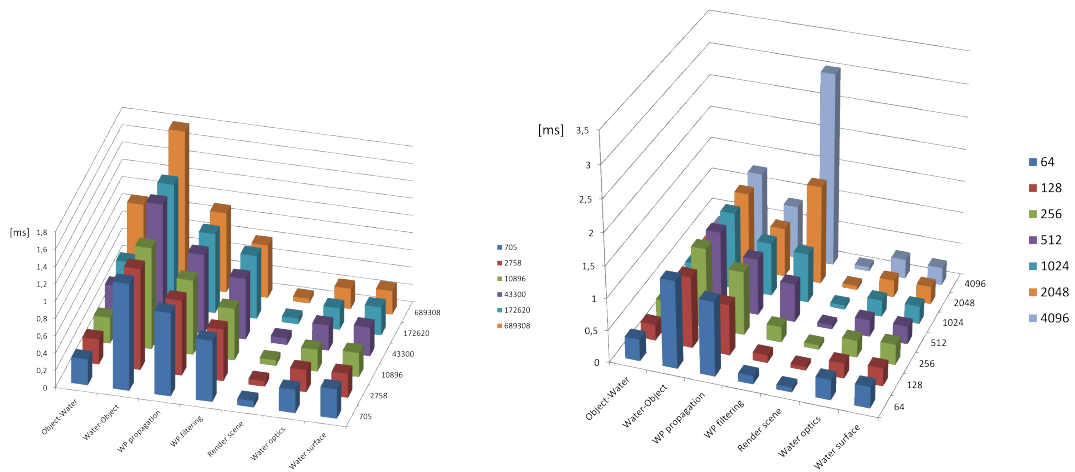


Figure 6.7: Mesh complexity influence on the simulation performance (left). Variable part represents number of object faces. Particle texture resolution complexity (right).



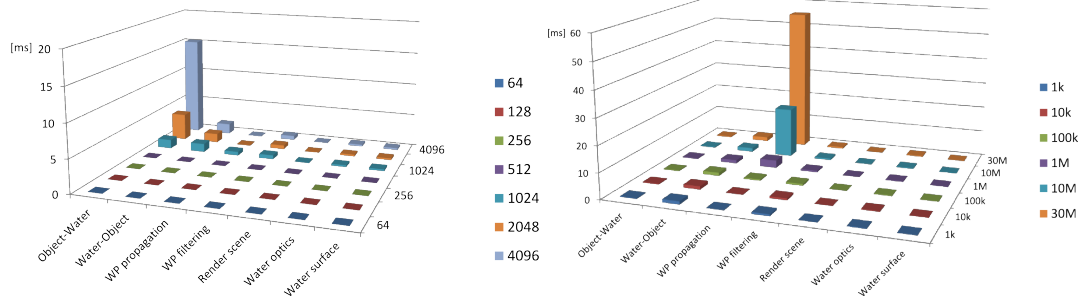


Figure 6.8: Influence of the silhouette texture resolution in different algorithm stages (left). Performance testing with respect to the number of wave particle (right).

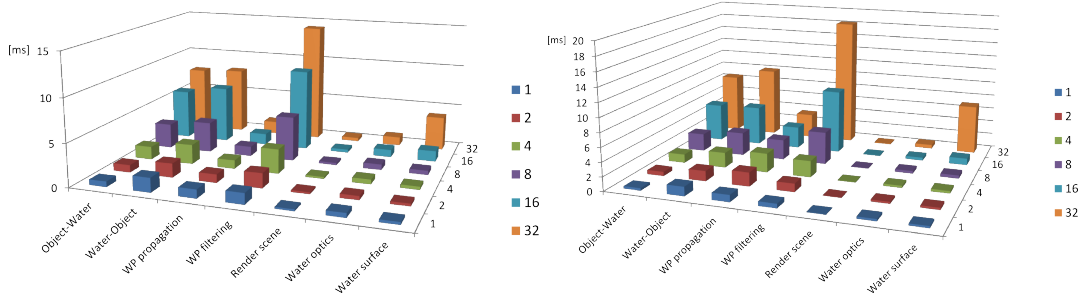


Figure 6.9: Result of the test case C (left) and case D (right).



## Chapter 7

# Conclusion

This thesis addresses simulating a virtual environment with large bodies of water. We presented a theoretical background and described various water phenomena, which occur in real life. In the field of fluid simulation we focus the real-time methods and we have described techniques which are used in high frame rate applications such as video games.

We have implemented the wave particle method developed by Yuksel et al. [YHK07]. Moreover, we described the process of a handling interaction between water body and a rigid body and implemented a water simulation with user interaction. We have showed that the simulating can produce plausible results and that it can be used at interactive frame rates. Simulation scenario have been created to show the capabilities of this method.

Even though we first addresses the method to work with arbitrary shaped objects, we have described some limitations for the use of a rigid body in our simulation.

Consequently, we have created water rendering module which offers an approximative solution to light reflection and refraction. Adaptive tessellation had been implemented to save the performance and further increase the speed of surface rendering.

### 7.1 Future work

Interesting improvement of the wave particle method would be handling the diffraction effect. This would allow to use this method extensively in the scenarios with local boundaries. As can be seen on the pool scenario, it is hard to comprehend the shape of the simulated water without seeing the water caustics. Besides caustics, there are localized water phenomena which could be mapped onto wave particle method. For example extracting the water white caps from existing high velocity particle. Moreover, adding breaking waves factor to wave particle method is a great opportunity for improvement.



# Bibliography

- [BMF07] Robert Bridson and Matthias Müller-Fischer. Fluid simulation: Siggraph 2007 course notes. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, pages 1–81, New York, NY, USA, 2007. ACM.
- [CF07] CT Chou and LC Fu. Ships on real-time rendering dynamic ocean applied in 6-dof platform motion simulator. In *CACS International Conference*, volume 3, 2007.
- [CLW<sup>+</sup>07] Haogang Chen, Qicheng Li, Guoping Wang, Feng Zhou, Xiaohui Tang, and Kun Yang. An efficient method for real-time ocean simulation. In Kin-chuen Hui, Zhigeng Pan, RonaldChi-kit Chung, CharlieC.L. Wang, Xiaogang Jin, Stefan Göbel, and EricC.-L. Li, editors, *Technologies for E-Learning and Digital Entertainment*, volume 4469 of *Lecture Notes in Computer Science*, pages 3–11. Springer Berlin Heidelberg, 2007.
- [CM10] Nuttapon Chentanez and Matthias Müller. Real-time simulation of large bodies of water with small scale details. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 197–206, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.
- [DB12] Jonathan Dupuy and Eric Bruneton. Real-time animation and rendering of ocean whitecaps. In *SIGGRAPH Asia 2012 Technical Briefs*, SA '12, pages 15:1–15:3, New York, NY, USA, 2012. ACM.
- [DCGG11] Emmanuelle Darles, Benoît Crespin, Djamchid Ghazanfarpour, and Jean-Christophe Gonzato. A survey of ocean simulation and rendering techniques in computer graphics. *CoRR*, abs/1109.6494, 2011.
- [FOK05] Bryan E. Feldman, James F. O'Brien, and Bryan M. Klingner. Animating gases with hybrid meshes. *ACM Trans. Graph.*, 24(3):904–909, July 2005.
- [Hav] Vlastimil Havran. Lecture notes for course realistic image synthesis.
- [IGLF06] Geoffrey Irving, Eran Guendelman, Frank Losasso, and Ronald Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Trans. Graph.*, 25(3):805–811, July 2006.
- [Lig] Lighthouse3D. Opengl timer query.

- [Max81] Nelson L. Max. Vectorized procedural models for natural terrain: Waves and islands in the sunset. *SIGGRAPH Comput. Graph.*, 15(3):317–324, August 1981.
- [MMS04] Viorel Mihalef, Dimitris Metaxas, and Mark Sussman. Animation and control of breaking waves. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '04, pages 315–324, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.
- [NAS10] NASA. The drag equation, 2010. Online: <http://www.grc.nasa.gov/WWW/k-12/airplane/drageq.html> accessed on Dec 19, 2014.
- [OCv13] Juraj Onderik, Michal Chládek, and Roman Ďurikovič. Sph with small scale details and improved surface reconstruction. In *Proceedings of the 27th Spring Conference on Computer Graphics*, SCCG '11, pages 29–36, New York, NY, USA, 2013. ACM.
- [Ope14] OpenGL. specification of opengl version 4.40, 2014.
- [RDP05] Vincent Ross, Denis Dion, and Guy Potvin. Detailed analytical approach to the gaussian surface bidirectional reflectance distribution function specular component applied to the sea surface. *J. Opt. Soc. Am. A*, 22(11):2442–2453, Nov 2005.
- [RWT11] Karthik Raveendran, Chris Wojtan, and Greg Turk. Hybrid smoothed particle hydrodynamics. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pages 33–42, New York, NY, USA, 2011. ACM.
- [SB12] Hagit Schechter and Robert Bridson. Ghost sph for animating water. *ACM Trans. Graph.*, 31(4):61:1–61:8, July 2012.
- [Tes01] Jerry Tessendorf. Simulating ocean water. 2001.
- [Tes04] Jerry Tessendorf. Interactive water surfaces. *Game Programming Gems 4, Charles River Media*, 2004.
- [TLP06] Adrien Treuille, Andrew Lewis, and Zoran Popović. Model reduction for real-time fluids. *ACM Trans. Graph.*, 25(3):826–834, July 2006.
- [TMFSG07] Nils Thürey, Matthias Müller-Fischer, Simon Schirm, and Markus Gross. Real-time breaking waves for shallow water simulations. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, PG '07, pages 39–46, Washington, DC, USA, 2007. IEEE Computer Society.
- [TRS06] Nils Thürey, Ulrich Rüdè, and Marc Stamminger. Animation of open water phenomena with coupled shallow water and free surface simulations. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '06, pages 157–164, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

- [Wei] Eric Weisstein. Method of shells. *MathWorld - A Wolfram Web Resource*.
- [YHK07] Cem Yuksel, Donald H. House, and John Keyser. Wave particles. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3), 2007.
- [YK09] Cem Yuksel and John Keyser. Fast real-time caustics from height fields. *The Visual Computer (Proceedings of CGI 2009)*, 25(5-7):559–564, 2009.
- [YNBH09] Qizhi Yu, Fabrice Neyret, Eric Bruneton, and Nicolas Holzschuch. Scalable real-time animation of rivers. *Computer Graphics Forum (Proceedings of Eurographics 2009)*, 28(2), mar 2009.
- [YNS11] Qizhi Yu, Fabrice Neyret, and Anthony Steed. Feature-based vector simulation of water waves. *Computer Animation and Virtual Worlds*, 22(2-3):91–98, 2011.
- [You81] A. T. Young. Rayleigh scattering. *Applied optics*, 1981.
- [Yuk10] Cem Yuksel. *Real-time Water Waves with Wave Particles*. PhD thesis, Texas A&M University, 2010.





# List of Figures

1.1	Results of our application. . . . .	2
2.1	Illustration of Lagrangian fluid element forming 2D wave. . . . .	5
2.2	Illustration of Eulerian fluid cell forming 2D wave. . . . .	6
2.3	Model of Gerstner wave in 2D. Plausible result (left), the case of exaggerated self intersection artefact (right). Green line shows original wave without vertical deviation, sharp red line shows a wave after vertical deviation, and the blue dotted line shows the vertical deviation. We put the vertical deviation in one image although the horizontal value is the deviation in vertical axis. . . . .	9
2.4	Model of Gerstner wave in 2D composed out of 3 components. Original wave (green), the final wave (red), and the vertical deviation (blue). . . . .	9
2.5	Frames taken from a water particle motion animation in deep waters. A water particle in a propagating wave in time (from left to right). The orange point is the initial position of the water particle. Current water particle position is marked by green point. . . . .	11
2.6	Frames taken from a water particle motion animation in shallow waters. . . . .	11
2.7	Illustration of the horizontal wave. Vertical component (left), horizontal component (centre), compounded wave (right). Courtesy of [YHK07]. . . . .	11
2.8	Wave diffraction after the wave passes the slit. Courtesy of [YHK07]. . . . .	12
2.9	Circular motion of under water flows in deep water (left) compared to the shallow waters (right). The motion of the water particles located deeper inside the water is reduced exponentially. . . . .	12
2.10	Reflectivity for a smooth water. Difference between the horizontally and vertically polarized component (left). Comparison of Schlick's approximation (right). . . . .	16
3.1	Modelling a wave using superposition principle. The red wave is the superposition of the blue and green one. . . . .	21
3.2	Illustration of box function. The grey sine wave is the original signal . . . . .	23
3.3	Superposition of the particle deviation function. The red line represents the square function, grey sine wave represents the original waveform function without and the blue is the . . . . .	23

3.4	Illustration of the wave front generation. Top images show the position of the wave particles and its radius $r_i$ (green circles). The bottom images show the global deviation function combined from local deviation function of these particles. . . . .	24
3.5	Illustration of wave fronts types: expanding wave front (left) and contracting wave front (right). Arrows represent the propagation direction. . . . .	25
3.6	Illustration of the wave front generation from particles. Source particles (left) and generated wave front (right) from the top view. . . . .	26
3.7	Illustration of the horizontal and vertical deviation function of a propagating wave. . . . .	26
3.8	Illustration of the influence of the horizontal deviation on a existing wave particle. Wave produced by a single wave particle without the effect of the horizontal deviation (left). Other images (centre, right) show a case of different vertical deviation strength $s_v$ . . . . .	27
3.9	Properties of the $i$ -th wave particle (blue circle) demonstrated from top 2D view. Yellow sector represents the dispersion angle $\delta$ , red sector is the propagation angle $\alpha$ , $\mathbf{x}$ is the current position, $\mathbf{o}$ is the origin and $r_i$ is the wave particle radius. . . . .	27
3.10	Illustration of the wave particle (dark blue circles) origin in the case of contracting (left) and expanding (right) wave front. The arrows represent the propagation angle, $\mathbf{o}$ is the origin point, and the light blue dotted circles represent the wave particle in an initial position. . . . .	28
3.11	Illustration of the wave particle (blue circles) behaviour without subdivision (left) and with subdivision (right) in three time steps. The opacity of the wave particles colour illustrates amplitude. Yellow sector on the right image represents the dispersion angle $\delta$ of the original (leftmost) particle. Note that the distances between particles should be even lower to form a continuous wave front; the particles are placed sparsely to increase readability. The difference in time $t_0$ is in both images is that there are all three particles placed in the same point on the left image. On the other hand there is only one particle which will be subdivided in the right image . . . . .	29
3.12	Illustration of the dispersion angle partitioning after particle subdivision operation. Particles are marked by blue circles with a unique identifier. Dispersion angle of each particle is marked by colour to enhance lucidity. . . . .	30
3.13	Visualization of the distances used by the particle subdivision method. The blue circle is the wave particle and the yellow region is the dispersion angle $\delta$ in the time of particle creation when the distance travelled $r_\delta = 0$ . . . . .	31
3.14	Exaggerated effect of the approximation error presented by the radial definition of the wave particle deviation function. Images are ordered by $w$ . Wave front with no error (centre), wave front with undesirable valleys (left) and peaks (right). . . . .	32

3.15	Illustration of the approximation error with respect to the distance between adjacent particles $w$ and the distance travelled $r_\delta$ by a particle. $T_w$ refers to the subdivision threshold. The red line represents the approximation error. The green and blue line is the upper, respectively lower, bound of the error. . . . .	32
3.16	Illustration of the wave particle reflection. In this case $r_\delta$ is wrongly interpreted as it does not cover the distance which has been travelled by the current particle. . . . .	34
3.17	Illustration of the wave particle (blue circle) reflection off of the flat water boundary (left) and curved boundary (right). The grey area represents the boundary with the mirrored origin. The dotted sector represents the dispersion angle before reflection and the yellow sector is the dispersion angle after reflection. . . . .	34
3.18	Demonstration of a wave front (blue line) approaching corner of water boundary in time (from left to right). Red arrows represent the propagation angle of the wave front. Valid reflection with the right angle corner (top) and reflection from an obtuse angled corner (bottom). . . . .	35
3.19	Different cases of wave propagation with respect to the position and the motion of the floating object. Striped line represents the object position in the previous time step. Cases O1, O2 (outside) show the influence of only the indirect wave effect since the object is on the water surface level. Cases I1 and I2 (inside) show the influence of both direct and indirect wave effect inside the volume. . . . .	38
4.1	Simplified OpenGL pipeline. Yellow coloured boxes represents the programmable stages, blue are the fixed stages, and violet represents output buffers. . . . .	44
5.1	Illustration of Transform feedback buffer swapping. The output of first TFB is used as an input of the second TFB. . . . .	46
5.2	Wave particle structure encoded for the use on the GPU. . . . .	49
5.3	Illustration of the data organization in the particle generator vertex buffer. . . . .	52
5.4	Illustration of a prone state of wave particle subdivision in the environment with water body boundary. . . . .	53
5.5	Illustration of splatting (left) and gathering (right) approach. . . . .	53
5.6	. . . . .	55
5.7	Red area represents the silhouette of the floating object. . . . .	57
5.8	Upper image row represents a water tank seen from side view. We show different positions of a sphere relatively to the water level. Note that the sphere is green on the top (positive z-coordinate of the normal) and red on the bottom (negative normal). The bottom row shows the same object seen from a top view without the part which is above the water level. The silhouette is the union of the green and red part while the contour is outer border of the red part. . . . .	58

5.9	Illustration of the contour generation. Object (above) is not monotonous by z-axis a forms an prone contour. The object bellow forms a valid contour. The arrows represent 2D normals of to contour. . . . .	59
6.1	Simulation environment rendered in our application . . . . .	64
6.2	Comparison of wave propagation of real life scene with a simulated scenario. . . . .	65
6.3	Video sequence of real wave propagation in a water tank. . . . .	65
6.4	Video sequence of simulated wave propagation in a pool. . . . .	66
6.5	Video sequence of the floating boat in lighthouse scene. . . . .	67
6.6	Video sequence of the boat interaction in lighthouse scene. . . . .	68
6.7	Mesh complexity influence on the simulation performance (left). Variable part represents number of object faces. Particle texture resolution complexity (right). . . . .	68
6.8	Influence of the silhouette texture resolution in different algorithm stages (left). Performance testing with respect to the number of wave particle (right). . . . .	69
6.9	Result of the test case C (left) and case D (right). . . . .	69

# List of Tables

4.1	List of dependencies of our application and their versions. . . . .	43
6.1	Evaluation of the approximative volume computation method used in our simulation. . . . .	63
6.2	Average frame rate for the test case C and D. . . . .	63



# List of source codes

5.1	Initialization of the Transform Feedback Buffer. . . . .	48
5.2	Data packing of the wave particle size . . . . .	49
5.3	Main part of the wave particle propagation program on vertex shader . . . . .	50
5.4	Specification of input and output primitives in geometry shader. . . . .	51
5.5	Geometry shader implementation of subdivision procedure. . . . .	51
5.6	Procedure of generating new particles on geometry shader. . . . .	52
5.7	Volume computation on the fragment shader. . . . .	55
5.8	Wave effect categorization of individual faces on vertex shader. . . . .	57
5.9	Recognizing the silhouette contour pixel. . . . .	57
6.1	OpenGL asynchronous time measuring. . . . .	62





# Appendix A

## List of abbreviations

<b>API</b>	Application Programming Interface
<b>CPU</b>	Central Processing Unit
<b>GPU</b>	Graphic Processing Unit
<b>GUI</b>	Graphics User Interface
<b>OpenGL</b>	Open Graphics Library
<b>GLEW</b>	OpenGL Extension Wrangler Library
<b>GLSL</b>	OpenGL Shading Language
<b>GLM</b>	OpenGL Mathematics
<b>NSE</b>	Navier-Stokes Equation
<b>VBO</b>	Vertex Buffer Object
<b>TFB</b>	Transform Feedback Buffer
<b>TF</b>	Transform Feedback
<b>SBT</b>	Solid Boundary Treatment
<b>SPH</b>	Smoothed Particle Hydrodynamics
<b>BRDF</b>	Bidirectional Reflectance Distribution Function



# Appendix B

## User manual

When running the application from the command line, it takes 3 arguments.

Part of the distribution is configuration directory where are the necessary information for the simulation and rendering phase. Most of these parameters are also accessible via GUI. Although parameters, which are set in the GUI, are not persistent to the next application run.

**path** Relative path to the configuration file, which sets simulation parameters.

**isTest** Second parameter is boolean flag (1/0) whether to use testing loop or the standard rendering loop.

**scene** Number of of the current testing scenario.

### B.1 Controls

User input can be also done by keyboard. The most important keys are as follows:

**F** Follow mode. Fix the camera to the current floating object (toggle)

**R** Ride mode. Redirect the effect of the WSAD keys to control the ship instead of camera (toggle)

**W** Move forward

**S** Move backwards

**D** Move right

**A** Move left

**K** Reset the the active floating object position

**Z** Change the active floating object to the next one

**P** Pause the wave particle simulation

**I** Display textures

**O** Orthographic view

# Appendix C

## DVD content

```
/
├── bin
├── videos
├── resources
│   ├── config
│   ├── images
│   ├── models
│   └── shaders
├── docs
│   ├── doxygen
│   └── thesis
├── src
├── lib
│   ├── AntTweakBar
│   ├── assimp
│   ├── DevIL-SDK-x86-1.7.8
│   ├── dirent
│   ├── glew-1.11.0
│   ├── glfw-3.0.4.bin.WIN32
│   ├── glm
│   ├── inih
│   ├── lodepng
│   ├── noise
│   └── rapidjson
└── vsproj
```