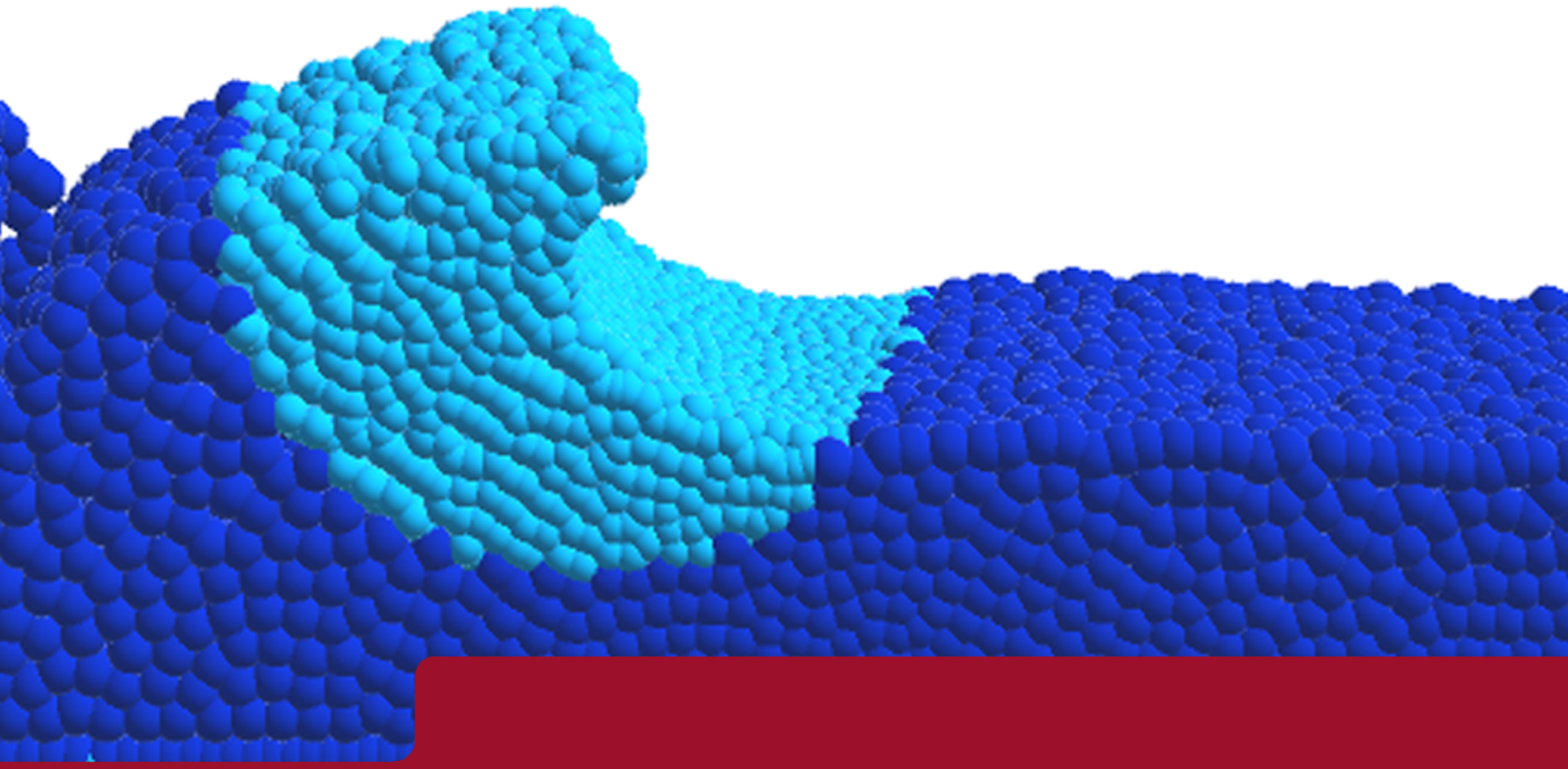
DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2020

An Interactive Eye-tracking based Adaptive Lagrangian Water Simulation Using Smoothed Particle Hydrodynamics

AJLA ELMASDOTTER



KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

An Interactive Eye-tracking based Adaptive Lagrangian Water Simulation Using Smoothed Particle Hydrodynamics

AJLA ELMASDOTTER

Master in Computer Science
Date: June 2020
Supervisor: Johan Hoffman
Examiner: Tino Weinkauf
School of Electrical Engineering and Computer Science
Swedish title: En interaktiv och adaptiv vattensimulering med användning av eye-tracking och Smoothed Particle Hydrodynamics

Abstract

Many water animations and simulations usually depend on time consuming algorithms that create realistic water movement and visualization. However, the intrigue for realistic, real-time and interactive simulations is steadily growing for, among others, the game and Virtual Reality industry. A common method used for particle based water simulations is the Smoothed Particle Hydrodynamics, which also allows for refinement and adaptivity that focuses the computational power on the parts of the simulation that require it the most.

This study suggests an eye-tracking based adaptive method for water simulations using Smoothed Particle Hydrodynamics, which is based on where a user is looking, with the assumption that what a user cannot see nor perceive is not of a greater importance. Its performance is evaluated by comparing the suggested method to a surface based adaptive method, by measuring frames per second, the amount of particles in the simulation, and the execution time . It is concluded that the eye-tracking based adaptive method performs better than the surface based adaptive method in four out of five scenarios and should hence be considered a method to further evaluate and possibly use when creating applications or simulations requiring real-time water simulations, with the restriction that eye-tracking hardware would be necessary for the method to work.

Sammanfattning

Flertalet vattensimuleringar samt animeringar brukar ofta vara beroende av tidskrävande algoritmer som skapar realistiskt utséende och realistiska rörelser. Däremot har intresset för realistiska, interaktiva realtidssimuleringar och liknande applikationer börjat växa inom, bland annat, spel- och virtual-realityindustrin. Smoothed Particle Hydrodynamics är en vanlig metod som används inom partikelbaserade vattensimuleringar, som även tillåter adaptivitet vilket fokuserar resurserna i datorn på de delar av simuleringen som kräver dem mest.

Denna studie föreslår en eye-trackingbaserad adaptiv metod för vattensimuleringar som använder sig av Smoothed Particle Hydrodynamics, som fokuserar adaptiviteten där användaren tittar i simuleringen med antagandet att det en användare inte kan uppfatta eller se inte är av relevans. Metodens prestanda evalueras genom jämförelse mot en adaptiv method som fokuserar adaptiviteten på vattnets yta och objekt runt vattnet, genom att mäta antalet renderade bilder per sekund, antalet partiklar i simulationen, samt exikveringstiden. Slutsatsen är att den eye-trackingbaserade adaptiva metoden presterar bättre än metoden som fokuserar adaptiviteten på vattnets yta i fyra av fem scenarion, och bör därför ses som en metod som har potential att utforskas vidare samt en metod som kan användas vi realtidssimuleringar av vatten, med begränsningen att hårdvara för eye-tracking behövs.

Contents

Chapter 1

Introduction

The topic of fluid simulations is always evolving and new methods and approaches are developed to handle the demand of increasingly realistic simulations. Commonly simulations are done offline which can produce highly realistic fluid behaviour as well as high resolution images, satisfying the needs of industries such as animation and movies. These simulations however can sometimes take hours if not days to produce an end result. Hence it is increasingly common to conduct real-time fluid simulations, which is also one of the more challenging problems in regards to fluid simulations.

Most algorithms developed for fluid simulation are based on different methods derived from the Navier-Stokes equations, which describes fluid motion, making it possible to create accurate and realistic fluid movement [Lau13]. Fluid simulations are created by solving the equations using different methods, depending on the approach used. More accurately solving the equation, minimizing the errors produced, creates a more realistic movement of the fluid. However the increased realism also implies increased computational power, possibly making it unviable to render the fluid in real-time.

There are mainly two different approaches to simulating fluid: The Eulerian method as well as the Langrangian method. The Eulerian method is a so called grid-based method, encapsulating the fluid within a boundary and simulates the behaviour of the fluid within the boundary. The Lagrangian method is a particle based method and instead allows for a boundary free implementation. For real-time, interactive, fluid simulation it can be argued that the Lagrangian method is preferable as it is cheaper to compute the simulation.

Smoothed particle hydrodynamics (SPH) is one commonly used Lagrangian method for fluid simulation. It was originally created to simulate astrophysical phenomena [Luc77; GM82], but has later been adapted and used for, among other, various fluid simulations [MCG03]. As the SPH method is a particle based method, it allows for mesh free simulations, making particle specific computation possible and easily adjustable for various variables such as density, pressure forces, etc. It is also argued to be one of the better methods for a Lagrangian fluid simulation model [He+10].

An adaptive fluid simulation implies that refinement of the grid or of the particles is executed depending on different conditions, such as distance to surface, interaction with objects, etc [He+10; ATW13], while some also include what is visible to the camera in the algorithm to further refine the simulation [SG11]. The adaptiveness allows for more complex simulations, while reducing the computational cost as fewer grid points or particles are needed in the simulation. Because of the benefits, adaptive algorithms are a common research topic both for offline simulations as well as real-time simulations.

Eye-tracking is becoming an increasingly common hardware used in, among other thing, virtual reality head mounted displays where it can be used to optimize rendering by focusing on the point where the user is looking [Gue+12]. The optimization of the rendering is based on the fact that the eye cannot perceive everything it sees. Instead, details are lost as the object moves further away from the so called view point of the user. It can be argued that a fluid simulation can utilize this same idea to provide more details and a higher resolution where the user is looking, which has been done using view dependant algorithms, see section 2.4, but not using eye-tracking. This thesis hence aims to explore the option of using an eye-tracking based adaptive algorithm for a fluid simulation.

1.1 Research Question

While this thesis aims to explore the option of using an eye-tracking based adaptive algorithm for a fluid simulation, it more specifically aims to measure the performance of the algorithm comparing it to a numerical counterpart. Hence the specific research question this thesis aims to answer is

- How effective is an eye-tracking based adaptive algorithm compared to a numerical counterpart when creating a Lagrangian water simulation using Smoothed Particle Hydrodynamics?

The performance is important when working with adaptive algorithms as the purpose

of them is to provide greater detail and resolution while maintaining or improving the performance of the algorithm.

1.2 Scope and Limitations

The basis for the eye-tracking based adaptive algorithm is that what cannot be seen or perceived does not need to be of such a high resolution or detail, as argued for in the case of rendering images [Gue+12]. Due to arguing that the surface and free surface flow, essentially the part of the simulation that is more visible, needs to be taken into account when creating an adaptive algorithm, a surface based adaptive algorithm has been chosen to be the numerical counterpart to the eye-tracking based adaptive algorithm. More specifically the algorithm suggested by Yan et al. [Yan+09] and He et al. [He+10] are going to be used to compare against the suggested eye-tracking based adaptive algorithm.

A fluid can both imply gas and liquid, where both can have varying implications in the implementation. For example, gases have a buoyancy which liquids might lack. This thesis will not be implementing any support for gaseous fluids and will instead only simulate water. The implementation for this thesis is based on work that can potentially also support gases and other liquids, however if one wishes to know more about those particular implementations, it is recommended to read the referenced papers.

An important aspect of an interactive fluid simulation, which this thesis will be implementing to answer the research question, is that the visualization is perceived to be realistic. As the objective of this thesis is to measure the performance of an eye-tracking based method, the perceived realism will instead be assumed and left as future work to further confirm it. However, the basis for this limitation are the papers which have compared water simulations made using Smoothed Particle Hydrodynamics to actual water behaviour [Del+09; Bot+10; Mon12] as well as user studies [UHT17], which confirm that it is a suitable and perceivably realistic method for water simulations. As the eye-tracking based adaptive algorithm is made using Smoothed Particle Hydrodynamics, it is assumed that the results applies for this algorithm as well, even though it is encouraged to conduct research confirming or rejecting this statement.

The last limitation of this thesis is that it is implemented using Unity v.2019.2.10f1 combined with HLSL. Hence no rendering system will be made from scratch, instead utilizng what Unity already has to offer. It also implies that there will be no

effort implementing a surface reconstruction algorithm, which would otherwise allow for a water simulation where a mesh-like construction would be used to visualize the simulation without showing any particles, however still using the particles as a basis for how the surface should look.

1.3 Thesis Outline

Chapter 2 presents necessary background for the thesis as well as relevant previous research within the different fields relating to this thesis. Chapter 3 presents the implementation of the basic Smoothed Particle Hydrodynamics framework as well as the adaptive frameworks used for this thesis. Chapter 4 then proceeds to explain how the research question is evaluated. Chapter 5 presents the results of the evaluation while chapter 6 discusses the results, limitations, and improvements as well as potential future work. Chapter 7 presents the conclusion and the answer to the research question. Appendix A contains figures showing the visualization of the water flow using both adaptive algorithms for the different scenarios used in the evaluation.

Keywords

SPH, eye-tracking, adaptive simulation, water simulation

Chapter 2

Background

This section aims to provide background for the thesis, including explaining the underlying concepts of the theory and the methods used. Furthermore this section will also present previous research done within the field of the thesis.

2.1 Fluid Simulations

A fluid refers to either a gas or a liquid, where the basic physical quantities of an isothermal, viscous fluid are mass-density ρ, pressure p, and velocity $\mathbf{u}$, which are all considered continuous fields in the fluid. These fields are governed by the Navier-Stokes Equations, which are often used in fluid simulations to reproduce fluid motion [MCG03]. With respect to a fixed coordinate system, the equation can be formulated as

$$\rho(\frac{\delta}{\delta t} + \mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \mu \nabla \cdot (\nabla \mathbf{u}) + \mathbf{f} \tag{2.1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2.2}$$

where μ is the dynamic viscosity of the fluid and $\mathbf{f}$ is the sum of external volume forces applied to the fluid. The second equation ensures that the flow is divergence free and hence also incompressible [Kel06; HHK08]. There exists mainly two methods to simulate fluid flow based on the Navier-Stokes Equations, the Eulerian method and the Lagrangian method.

2.1.1 The Eulerian Method

The Eulerian method is based on equations (2.1) - (2.2) for which the spatial domain is represented by a grid and the fluid is thought of being composed of fluid cells in

the grid, which is depicted in figure 2.1. Foster and Metaxas [FM96] are stated to be the first in the field of computer graphics to create a full 3D fluid simulation based on the Navier-Stokes equations. The simulation method was Eulerian and was based on the work of Harlow and Welch [HW65], the Marker and Cell method, which stores velocity components at the cell faces and pressure at the cell centers, resulting in a discretization with good stability and accuracy [ATW13]. An improvement for the stability was later introduced by Stam [Sta99] using a semi-Lagrangian scheme, allowing among others larger time steps, and has been further extended in various papers.

Although the method often provides high visual accuracy [Raj11] and also better describes some of the properties of the fluid, such as mass-density and pressure [Kel06], a disadvantage of the method is the grid itself. As the fluid is constrained within the grid, it cannot exist outside it and thus it can be expensive to create larger and complex simulations as well as uniform grids becoming more expensive for high resolutions [ATW13]. There exists implementations of adaptive grid methods to overcome these problems, such as the one suggested by Losasso, Gibou, and Fedkiw [LGF04] where an octree data structure is used to adaptively simulate water and smoke with fine detail, which has later been extended upon by, among others, Batty, Xenos, and Houston [BXH10] to further improve the speed of the simulation while maintaining the detail.

Another disadvantage of the Eulerian method is the computation time required to create the simulation. Depending on the complexity of the simulation, one frame can take multiple minutes to compute [ATW13; Kel06; LGF04], making it unviable for interactive real-time simulations. Wu et al. [Wu+18] does however suggest a significantly faster method which efficiently creates sparse grids for the fluid each frame, resulting in frames only requiring a few hundred milliseconds to be computed.

2.1.2 The Lagrangian Method

In the Lagrangian method the fluid is approximated by a set of particles, depcited in figure 2.1, which move according to the fluid velocity, where each particle carries parts of the information required to solve the system, such as local mass, position and velocity [Kel06] while the inter-particle forces, such as viscosity and pressure, are computed using kernels [HHK08]. Even though the Eulerian method has improved, allowing for changing grids [Wu+18], the Lagrangian method makes it possible for the fluid to move freely, as there is no grid limiting the fluid. While the method does require more computation with a larger amount of particles, it also allows for

an interactive solution as the computation is generally cheaper.

As the fluid is defined by its particles, each field quantity for the fluid depends only on time, t, while the acceleration of a particle is defined as the derivative of its velocity. The Navier-Stokes partial differential equations are, therefore, reduced to a system of ordinary differential equations for each particle, which allows for the following formulation of Navier-Stokes Equation for the Lagrangian method, with respect to a coordinate system that moves with the velocity of the flow,

$$\rho \frac{d\mathbf{u}}{dt} = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f} \tag{2.3}$$

As the right hand side of equation (2.3) consists of internal and external force fields, which can be combined into a sum $\mathbf{F}$, it follows that for a particle i the acceleration can be approximated by

$$\mathbf{a}_i = \frac{d\mathbf{u}_i}{dt} = \frac{\mathbf{F}_i}{\rho_i} \tag{2.4}$$

where $\mathbf{u}_i$ is the velocity for particle i, $\mathbf{a}_i$ is its acceleration, ρ_i is the local mass-density and $\mathbf{F}_i$ is the total force acting on the particle [MCG03].

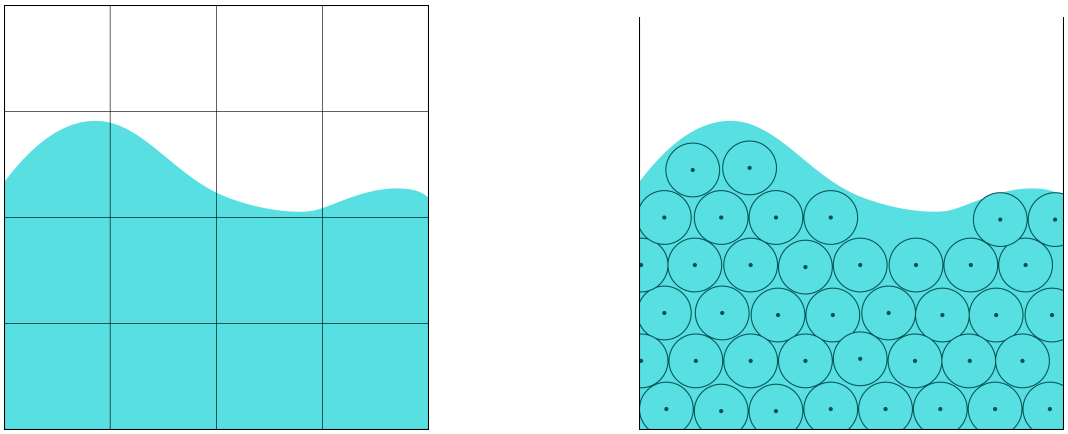


Figure 2.1: An Eulerian fluid is defined by cells (left) while a Lagrangian fluid is defined by its particles (right) .

2.2 Smoothed Particle Hydrodynamics

Smoother Particle Hydrodynamics (SPH) is one of the more popular Lagrangian methods for fluid simulations and was first introduced in the computational astrophysical field, designed for compressible flow problems and used to simulate various astrophysical phenomena by Lucy [Luc77] as well as Gingold and Monaghan

[GM82]. Stam and Fiume [SF95] introduced SPH to the graphics community, depicting fire and other gaseous phenomena. This method was later extended to fluid simulations by Müller, Charypar, and Gross [MCG03] and has since been widely used.

To combat the problem of artifical compressible behaviour in liquids, as the SPH method originally calculates the pressure using the ideal gas equation due to it being designed to work with gas as well as fluid, Becker and Teschner [BT07] proposes simulating free surface flows using a weakly compressible SPH. However this results in large forces and hence limits the length of the time step greatly. Other approaches that have been proposed include the predictive-corrective incompressible method suggested by Solenthaler and Pajarola [SP09] as well as the position based approach suggested by Macklin and Müller [MM13]. To increase stability in the pressure in the presence boundaries and handle penetration issues, recent research introduces the concept of density maps which is a method to simulate boundaries with friction [KB17], the concept of using Pressure Boundaries which computes pressure values at the boundary using the pressure Poisson equation [Ban+18b], and recent studies also suggests strong coupling for fluid pressures and rigid-body velocities [Gis+19].

Due to the method's Lagrangian nature, the fluid is represented by particles, where particle i has position $\mathbf{r}_i$ and mass m_i, which the SPH method then uses to approximate quantities such as pressure and mass-density. The basis formulation for the SPH method as well as its gradient and the Laplacian is defined below, with the assumption that the quantity kernel is first and second order differentiable. The SPH interpolation depends on the neighboring particles, which are denoted j, and their values to calculate the quantities, denoted A.

$$A(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h) \tag{2.5}$$

$$\nabla A(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} \nabla W(\mathbf{r} - \mathbf{r}_j, h) \tag{2.6}$$

$$\nabla^2 A(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} \nabla^2 W(\mathbf{r} - \mathbf{r}_j, h) \tag{2.7}$$

where A_j is the value of the quantity A at position $\mathbf{r}_j$, W is the smoothing kernel, m_j is the mass, ρ_j is the mass-density, and h is the so called support radius for W.

2.3 Adaptive Algorithms

An adaptive algorithm within fluid simulations implies that particles are split and merged based on different conditions in the Lagrangian case, while in the Eulerian case it implies that the mesh is being refined and coarsened depending on different conditions. Desbrun and Cani [DC99] introduced splitting and merging of particles and the idea of individual time-stepping for each particle. Improvements to the splitting and merging was later suggested by Adams et al. [Ada+07], which enables an improved surface reconstruction method where particle positions are adjusted after splitting and merging to create smoother surfaces. This was later extended by Solenthaler and Gross [SG11] by introducing a two-scale method, which avoids splitting and merging altogether, allowing for simulations with large resolution differences without any larger stability issues. There has also been suggestions to improve the individual time-stepping for each particle suggested by Desbrun and Cani [DC99], such as the adaptive stepping scheme suggested by Ban et al. [Ban+18a].

By extending the non-uniform system suggested by Desbrun and Cani [DC99] and the surface tracking method proposed by Adams et al. [Ada+07], Yan et al. [Yan+09] propose a generalized distance field function, which allows fast detection of surface and fluid body particles, as well as new sampling rules for splitting and merging, resulting in faster simulations as the method focuses the computational power on the smaller particles at the fluid surface. This has later been improved by He et al. [He+10], where they introduce a modified version of the distance field function, which enables splitting around various boundaries and not only the surface, allowing further details around boundaries as well. Both papers suggest approaches to implementing their adaptive algorithms on the GPU to further speed up the implementations. As GPU based methods often rely on dense cell structures [Gre10] or structures based on linked lists [WRR18], which suffer from scaling problems as well as performance problems for adaptive methods, Winchenbach and Kolb [WK19] suggests a multi-level-memory structure for adaptive SPH simulations. They introduce a hash-map based sparse data structure, which reduces the memory requirements for adaptive methods and improves the performance without interfering with the simulation.

2.4 View-Dependent Refinement

Adapting meshes or refining particles can be done based on the view-point of the user or the camera resulting in, for example, a higher or lower level-of-detail or

resolution, which in combination with model simplification is common when optimizing rendering of complex geometric models [LE97; Hop97; JLL09]. The area which is visible on the screen, or the area which the camera can perceive is called the *viewing frustum*. As argued by Koh, Narain, and O'Brien [KNO14] the goal of the view-dependant algorithm is to preserve the details of a simulation, while avoiding the wasted effort of simulating what is not apparent to the viewer. An example [KNO14] of view-dependant mesh refinement can be seen in figure 2.2.

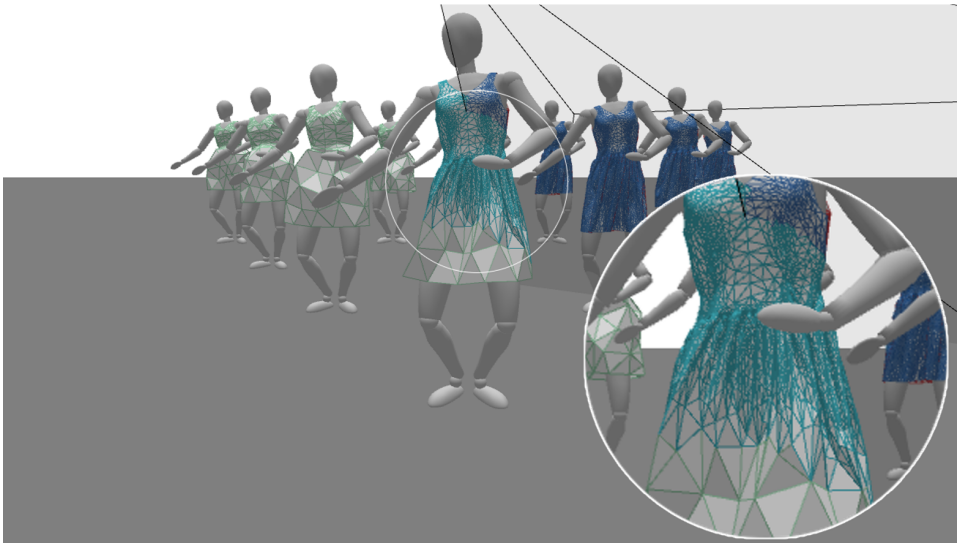


Figure 2.2: A finer mesh is seen within the viewing frustum of the camera, while a coarser mesh is used outside of it [KNO14].

Barran [Bar06] suggests a view-dependent fluid simulation taking into account the camera's coordinates in the world space when constructing the grid and hence modifying the resolution of the simulation based on the view-point of the camera. It does however suffer from instabilities as the cell symmetries can be broken by finer details [BK11]. Using octree-structures [LGF04], Kim, Ihm, and Cha [KIC06] and Bunlutangtum and Kanongchaiyos [BK11] both suggest adaptive view-dependent algorithms for mesh refinement, reducing the number of grid cells included in the computations while preserving the detail of the simulation in a more stable manner.

Two notable examples where the viewing frustum of the camera is taken into account for particle based simulations are Adams et al. [Ada+07] and Solenthaler and Gross [SG11]. Both papers include the viewpoint of the camera in the algorithm, deducing which parts of the simulation are not visible, allowing for faster simulations which also maintain the details of the fluid, while simultaneously reducing the

amount of particles needed for the simulation in total. The viewpoint of the camera allows the algorithms to deduce which parts of the simulation should have a higher resolution, as what is not visible for the camera can be set to have a lower resolution.

2.5 Eye-Tracking and Virtual Reality

While including the view-point in the view-dependant algorithms does reduce the required computation, a user will not be able to perceive the whole rendered picture due to the way the eye works [Gue+12] and it can hence be argued that it is a waste to allocate resources to compute fine details in the whole rendered picture. Instead, by tracking the user's gaze through the means of eye-tracking devices, it is possible to adapt the rendered picture and increase the resolution at the point which is visible for the user's gaze, while reducing it for the rest of the image [Poh+16]. Guenter et al. [Gue+12] is an example that utilizes eye-tracking to decrease the computational cost, basing the rendering of the image on the user's gaze.

These methods are also used to optimize rendering techniques used in *Virtual Reality* (VR) [Wei+16] which can be crucial to reach an acceptable frame rate that does not cause nausea. Creating interactive fluids has also increased in importance in VR prompting the research of optimization of fluid simulations specifically for VR [KS12; Hua+15]. However, as of writing this, there seems to be little to no contribution utilizing eye-tracking as an optimization method for interactive particle-based fluid simulations, neither in the field of Virtual Reality nor otherwise.

2.6 Contribution

While it can be argued that view-dependant adaptive algorithms are similar to utilizing eye-tracking, it is important to separate the two. Utilizing eye-tracking can potentially further reduce the computational cost of the simulation, as only the area within the user's gaze is refined. This paper hence aims to examine whether eye-tracking is a potential option, performance-wise as realism is assumed as discussed in section 1.2, for adaptive fluid simulations where Smoothed Particle Hydrodynamics is used as the simulation method to guarantee reaching an interactive performance. The eye-tracking based adaptive algorithm is also compared to a numerical counterpart, based on the work by Adams et al. [Ada+07], Yan et al. [Yan+09], and He et al. [He+10], to put the results in perspective to an already established method.

Chapter 3

Implementation

This section will describe the implementation, calculations and parameters used to evaluate the research question. Sections 3.1 through 3.5 will present the framework for a regular non-adaptive water simulation using SPH. Section 3.6 will present the framework for an adaptive simulation using SPH, including the specifics for this thesis. Finally, sections 3.7, 3.8, and 3.9 will present the GPU-implementation specifics, an overview of the simulation flow, and the parameters used respectively.

3.1 Smoothing Kernels

The accuracy and stability of the SPH method depends on smoothing kernels. The three kernels presented in this section are suggested by Müller, Charypar, and Gross [MCG03] and will be referenced throughout later sections.

As a default smoothing kernel for the simulation, a 6th degree polynomial kernel is suggested, which both preserves the Gaussian bell curve and is computationally cheaper than multiple counterparts [Kel06]. The kernel is defined as

$$W_{default}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - ||\mathbf{r}||^2)^3 & 0 \leq ||\mathbf{r}|| \leq h \\ 0 & ||\mathbf{r}|| > h \end{cases} \tag{3.1}$$

with the following gradient and Laplacian respectively

$$\nabla W_{default} = -\frac{945}{32\pi h^9}\mathbf{r}(h^2 - ||\mathbf{r}||^2)^2 \tag{3.2}$$

$$\nabla^2 W_{default} = -\frac{945}{32\pi h^9}(h^2 - ||\mathbf{r}||^2)(3h^2 - 7||\mathbf{r}||^2) \tag{3.3}$$

The default smoothing kernel is used to calculate most quantities, however as argued by Kelager [Kel06] if the gradient of the default kernel is used when calculating the pressure force, particles would clump together in regions with high pressure due to $\nabla W_{default} \to 0$ when $||\mathbf{r}|| \to 0$. To avoid these problems, the gradient of the spiky kernel is used as a smoothing kernel when calculating specifically the pressure force [MCG03]:

$$\nabla W_{pressure}(\mathbf{r}, h) = -\frac{45}{\pi h^6}\frac{\mathbf{r}}{||\mathbf{r}||}(h - ||\mathbf{r}||)^2 \tag{3.4}$$

To avoid adding energy to the simulation which potentially increases acceleration of particles, introducing instability to the simulation, the Laplacian for the viscosity force is constrained to be positive. This constraint also makes sure that the viscosity force acts as a damping force, reducing the velocity of particles with high energy [Raj11]. Hence, instead of the Laplacian of the default kernel, another Laplacian is used as a smoothing kernel for specifically the viscosity force [MCG03]:

$$\nabla^2 W_{viscosity}(\mathbf{r}, h) = \frac{45}{\pi h^6}(h - ||\mathbf{r}||) \tag{3.5}$$

3.2 Radii

For the implementation of the simulation there are two radii that need to be computed, namely the particle radius and the support radius. This section aims to highlight the difference as well as show how to calculate them.

3.2.1 Particle Radius

As each particle is considered to be a sphere, they also need a radius defining that sphere. Using the equation for the volume of a sphere

$$V = \frac{4}{3}\pi r^3 \tag{3.6}$$

it is possible to calculate the radius

$$r = \sqrt[3]{\frac{3m}{4\pi\rho_0}} \tag{3.7}$$

3.2.2 Support Radius

The support radius h for a smoothing kernel is an essential part to keep the simulation stable. A too large support radius causes the results to be inaccurate as particles

from far away in the simulation start affecting each other. A too small support radius however includes too few particles in the calculations and can cause instabilities. An illustration in 2D is shown in figure 3.1.

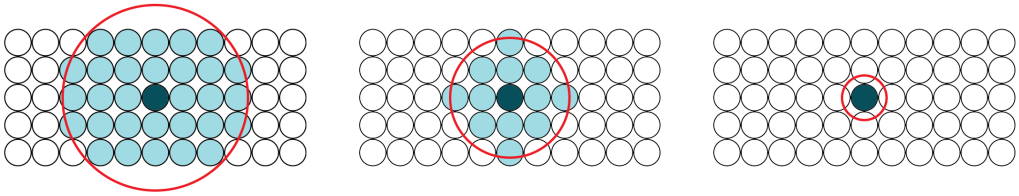


Figure 3.1: The support radius reach is marked in red, while the included particles are blue. The dark blue represents the focused particle. Using a too large support radius (*left*) can cause inaccuracies, while it being too small (*right*) causes instabilities. Enough particles (*middle*) provides stability to the simulation.

It is hence important to get a good enough support radius. The support radius is calculated using an average number of desired particles x that fits within the radius and covers the volume. The support radius h_i for a particle i can hence be computed using

$$h_i = \sqrt[3]{\frac{3V_i x}{4\pi}} \tag{3.8}$$

where V_i is the volume for particle i, which is computed using the rest density ρ_0, and x are the desired average number of kernel particles.

The support radius also poses as a constraint for which particles are considered neighboring particles. As all particles have the same support radius for the standard SPH framework, two particles are considered neighbors if the following holds

$$||\mathbf{r}_i - \mathbf{r}_j|| \leq h \tag{3.9}$$

3.3 Forces

To calculate the acceleration for each particle, the total force $\mathbf{F}_i$ is needed as seen in equation (2.4). The force consists of two components, the internal forces, which in turn consists of pressure forces and viscosity forces, as well as the external forces,

which include gravity and surface tension, and is for particle i defined as:

$$\mathbf{F}_i = \sum_n \mathbf{f}_i^n \tag{3.10}$$

where n represents the different forces, such as pressure force and surface tension.

This section will describe the general approach to compute these forces using SPH, while the modifications needed for an adaptive implementation will be described in later sections. Each individual force or quantity described in this section must be computed at each time step and is based on the work by Müller, Charypar, and Gross [MCG03] and Kelager [Kel06].

3.3.1 Mass-Density

The mass-density is a continuous field of the fluid, which must be computed and evaluated at the position of every particle. Using SPH, the density for a particle i is calculated as follows

$$\rho_i = \sum_j m_j W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{3.11}$$

where j denotes a neighboring particle, m_j is the mass and $\mathbf{r}_i$ is the position. Equation (3.1) is used as the smoothing kernel.

3.3.2 Pressure Forces

The pressure term used for a particle i was suggested by Desbrun and Gascuel [DG96], which is also used by Müller, Charypar, and Gross [MCG03], and is a modified version of the ideal gas state equation. It is defined as follows

$$p_i = k(\rho_i - \rho_0) \tag{3.12}$$

where k is the gas stiffness constant, ρ_i is the mass-density of particle i, while ρ_0 is the rest density for the fluid. When both density and the pressure term have been computed, the inter-particle pressure force is calculated using

$$\mathbf{f}_i^{\text{pressure}} = -\rho_i \sum_{j \neq i} (\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}) m_j \nabla W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{3.13}$$

where equation (3.4) is used as a smoothing kernel.

3.3.3 Viscosity Forces

A fluid's viscosity defines the flow's resistance, which is represented by the viscosity constant μ. The viscosity force acts as a damping force to the particles and decreases the kinetic energy. The force for a particle i is computed using

$$\mathbf{f}_i^{\text{viscosity}} = \mu \sum_{j \neq i} (\mathbf{u}_j - \mathbf{u}_i) \frac{m_j}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{3.14}$$

for each neighboring particle j, where $\mathbf{u}_i$ is the velocity for particle i and equation (3.5) is used as a smoothing kernel [MCG03].

3.3.4 Gravity

The gravity force acts equally upon the particles using

$$\mathbf{f}_i^{\text{gravity}} = \rho_i \mathbf{g} \tag{3.15}$$

where $\mathbf{g}$ is the downwards gravitational force.

3.3.5 Surface Tension

Within the fluid, the inter-molecular attraction is equal in all directions and the particles within the fluid are hence held in balance. For particles at the surface of the fluid however, this is not the case and the imbalance existing at these surface particles gives rise to the surface tension force [MCG03; Raj11]. To identify surface particles and where the surface tension force should be applied, a so called smoothed *color field* c_i is used

$$c_i(\mathbf{r}_i) = \sum_j \frac{m_j}{\rho_j} W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{3.16}$$

where equation (3.1) is the used smoothing kernel. The surface normal, which points inwards to the fluid, is given by the gradient of the smoothed color field and the divergence of the normal gives the curvature κ of the surface

$$\mathbf{n}_i = \nabla c_i \tag{3.17}$$

$$\kappa_i = \frac{-\nabla^2 c_i}{||\mathbf{n}_i||} \tag{3.18}$$

The surface force is then calculated using

$$\mathbf{f}_i^{\text{surface}} = \sigma \kappa_i \mathbf{n}_i = -\sigma \nabla^2 c_i \frac{\mathbf{n}_i}{||\mathbf{n}_i||} \tag{3.19}$$

where σ is the tension coefficient. The force is only applied to particles located at the surface of the liquid, which is determined by the length of the normal. If the length is above a certain threshold l, the surface tension is computed, otherwise it is considered 0. The threshold is set to be

$$l = \sqrt{\frac{\rho_0}{x}} \tag{3.20}$$

where x is the amount of kernel particles and ρ_0 is the rest density [Kel06].

3.4 Time Integration

Each particle is advanced using a fixed time step Δt, where its new position is computed by integrating the acceleration obtained by equation (2.4). For simplicity the semi-implicit Euler scheme is used

$$\mathbf{u}_{t+\Delta t} = \mathbf{u}_t + \Delta t \mathbf{a}_t \tag{3.21}$$

$$\mathbf{r}_{t+\Delta t} = \mathbf{r}_t + \Delta t \mathbf{u}_{t+\Delta t} \tag{3.22}$$

Notice that the updated velocity is utilized when predicting the new position.

3.5 Collision Handling

To hinder the fluid to phase through borders or objects in the simulation, collision handling is necessary. To detect a collision the following information is needed:

- The contact point **cp**
- The penetration depth d
- The surface normal **n**

These are calculated using primarily a particles position, but can utilize the velocity if needed be. The contact point is the point of impact, the penetration depth is the distance the particle has traveled inside the object or outside the boundaries, and the surface normal is a vector perpendicular to the surface of the collision object at the contact point, directed away from the object.

As this thesis is limited to only primitive types, tetrahedral meshes and similar is

not covered but can be read more about in [Kel06]. All collisions can be defined through the function F

$$\begin{array}{ll} F(\mathbf{x}) < 0 & \mathbf{x} \text{ is inside the primitive} \\ F(\mathbf{x}) = 0 & \mathbf{x} \text{ is on the surface of primitive} \\ F(\mathbf{x}) > 0 & \mathbf{x} \text{ is outside the primitive} \end{array} \tag{3.23}$$

The signum function is also used which is defined as

$$\text{sgn}(\mathbf{x}) = \begin{cases} -1 & x < 0 \\ 0 & x == 0 \\ 1 & x > 0 \end{cases} \tag{3.24}$$

3.5.1 Sphere

In combination with equation (3.23), a sphere can be implicitly defined by

$$F(\mathbf{x}) = ||\mathbf{x} - \mathbf{c}||^2 - r^2 \tag{3.25}$$

where **c** is the center of the sphere, r is the radius of the sphere and **x** is the position of the particle. When a collision occurs, the contact point for a sphere is defined by

$$\mathbf{cp} = \mathbf{c} + r\frac{\mathbf{x} - \mathbf{c}}{||\mathbf{x} - \mathbf{c}||} \tag{3.26}$$

the penetration depth is defined by

$$d = |||\mathbf{c} - \mathbf{x}|| - r| \tag{3.27}$$

with the surface normal being defined by

$$\mathbf{n} = \text{sgn}(F(\mathbf{x}))\frac{\mathbf{c} - \mathbf{x}}{||\mathbf{c} - \mathbf{x}||} \tag{3.28}$$

3.5.2 Cylinder

Collision with a cylinder is approached in a similar manner to a sphere, where the y-coordinate of the center point is instead varying. In this implementation only cylinders that go from the floor to the roof are utilized and hence the cylinders y-coordinate is always set to be equal to the y-coordinate of the particle. If one wishes to extend the collision implementation and utilize cylinders that do not reach the roof or the floor, one can define a maximum or minimum value for the y coordinate and also include this in the collision handling.

3.5.3 Boxes

Kelager [Kel06] describes the collision handling, including how to get the contact point, penetration depth and surface normal, of an oriented bounding box, which allows rotation of the box without affecting the collision handling. As the boxes in this thesis are mainly used as boundaries, rather than obstacles, the collision detection will be simplified compared to the suggestions by Kelager [Kel06] and further reading is hence referenced to that paper.

For each axis, a particles position is compared to the maximum value and the minimum value of the box on that axis. If the x-coordinate of a particle has a greater value than the maximum x-coordinate of the bounding box, a collision has taken place on the x-axis and the x-coordinate of the contact point is set to the x-coordinate of the maximum value of the bounding box.

$$\mathbf{cp}_x = \mathbf{r}_{\text{box-max}_x} \tag{3.29}$$

where the radius of the particle also needs to be taken into account. The collision for the other axes are computed in the same manner. Similarly a collision has taken place if a coordinate has a smaller value than the coordinate of the minimum value of the bounding box. The normal on each axis is computed by the difference in position on that axis:

$$\mathbf{n}_x = \mathbf{r}_{\text{box}_x} - \mathbf{r}_{i_x} \tag{3.30}$$

where the final normal $\mathbf{n}$ needs to be normalized.

3.5.4 Collision response

After a collision has been detected and the necessary information computed, a response to the collision is needed. For each fluid particle i that collides with an object, the position is updated using the contact point

$$r_i = \mathbf{cp} \tag{3.31}$$

while the velocity is updated according to

$$\mathbf{u}_i = \mathbf{u}_i - (\mathbf{u}_i \cdot \mathbf{n})\mathbf{n} \tag{3.32}$$

Observe that Kelager [Kel06] suggests a slightly modified variant of the velocity response, as it includes gases, which is not in the scope of this thesis.

3.6 The Adaptive Framework

In traditional SPH using a large amount of smaller particles to simulate larger animations requires a significant computational cost. By using varying sizes for the particles, it is possible to reduce the computational burden without reducing the realism [He+10]. For this thesis there are 2 different sizes that a particle can have; *LARGE* or *SMALL*. This section will describe the adaptive SPH framework used in this thesis, by introducing the changes needed to the traditional SPH framework and then explaining the two different adaptive methods used.

3.6.1 Modifications to the Traditional SPH Framework

In the adaptive framework, particles vary in size and thus different particles have different radii and different support radii. In the traditional SPH framework, two particles are considered neighbors if the following holds

$$||\mathbf{r}_i - \mathbf{r}_j|| \leq h \tag{3.33}$$

where h is the support radius. For the traditional SPH framework, the support radius is identical for all particles, which is not the case for the adaptive framework. In the adaptive framework two particles are instead considered neighbors if the following holds

$$||\mathbf{r}_i - \mathbf{r}_j|| \leq \frac{h_i + h_j}{2} \tag{3.34}$$

using the individual support radii for both particles as suggested by Yan et al. [Yan+09]. In a similar manner the smoothing kernels are also modified using

$$\frac{W(\mathbf{r}_i - \mathbf{r}_j, h_i) + W(\mathbf{r}_i - \mathbf{r}_j, h_j)}{2} \tag{3.35}$$

as suggested by Adams et al. [Ada+07] which also forces Newton's third law, where each action causes a reaction, for non-uniformly sized particles.

The number of kernel particles x can vary for the different support radii and hence the threshold for the computation of the surface tension needs modifications, as its computation depends on the amount of kernel particles. The threshold was set to depend only on the number of kernel particles for the larger sized particles, namely x_{LARGE}, as it provides a greater stability and is hence defined to be

$$l = \sqrt{\frac{\rho_0}{x_{LARGE}}} \tag{3.36}$$

3.6.2 Splitting and Merging Particles

The adaptive framework for this thesis is based on the splitting and merging of particles, which is a process that must preserve the mass [HHK08] of the particles as it otherwise removes volume from the fluid and introduces instabilities. When a particle meets certain conditions it should be split into smaller particles or combined into a larger particle. The method specific conditions for splitting and merging of particles are presented in later sections, however this section presents the conditions that are common for both adaptive methods used in this implementation.

Splitting

When splitting a particle, a set of new particles are generated that are placed within the radius of the parent particle. The mass of the parent particle is distributed equally among the child particles, while velocities are kept equal to the parent. Raising the original radius by an exponent of 3 negates the cubic root in equation (3.7), which in combination with equation (3.37) is used to calculate the new radius. The new values for a child particle j are hence given by

$$m_j = \frac{m_i}{n} \tag{3.37}$$

$$r_j = \sqrt[3]{\frac{r_i^3}{n}} \tag{3.38}$$

$$\mathbf{u}_j = \mathbf{u}_i \tag{3.39}$$

where n is the number of child particles generated. The number of generated child particles is arbitrary and has for this implementation been set to two for simplicity. Aside from the condition set by the adaptive method used, a particle i is split into two smaller particles if and only if the particle is larger than the smallest set size. This prevents particles of the smallest size from being split further than desired. As there are only two different sizes in this implementation, a particle is split if and only if it is of size *LARGE*. Additionally, the different values for mass and radius used after a split are pre-computed, as these values are constant and will not differ throughout the simulation.

Merging

As a particle is split into two child particles, for a merge to take place exactly two smaller particles are needed. The new particle mass is given by the sum of masses of the child particles that are merged, which the calculation for the new radius also utilizes in combination with equation (3.7), similarly to how it is done when splitting

particles. As each particle i included in the merging process is of the same size, the new mass, radius, position, and velocity of a merged particle j are in general terms given by the following

$$m_j = \sum_i m_i \tag{3.40}$$

$$r_j = \sqrt[3]{\sum_i r_i^3} \tag{3.41}$$

$$\mathbf{u}_j = \frac{\sum_i \mathbf{u}_i}{n} \tag{3.42}$$

$$\mathbf{r}_j = \frac{\sum_i \mathbf{r}_i}{n} \tag{3.43}$$

where n is set to two and defines the number of particles included in the merge. A group of particles are merged if and only if they are of a size smaller than the largest size, which in this implementation implies that the group is merged if and only if they all they all are of size *SMALL*, aside from the condition set by the used adaptive method. Similarly to splitting, the radius and mass used after a merge have been pre-computed instead of being calculated throughout the simulation.

3.6.3 Surface based Adaptive Framework

A numerical adaptive framework based on the surface of the flow as well as the existing boundaries implies that the surface and boundaries dictate the different sizes of the particles. A particle that belongs to the surface or is close to a boundary is smaller in size, while a particle belonging to the fluid body is larger, as suggested by He et al. [He+10]. An example is depicted in figure 3.2.

To decide which particles are considered close enough to the surface or to the boundaries and which particles belong to the fluid body, a *character field* function is used [He+10], which in turn considers both geometric details as well as physical complexity. There are three main aspects that the character field function consists of:

Firstly the geometric complexity. As the surface and the boundaries should get more focus, the particles close to the surface or a boundary have a higher level of detail and hence a smaller radius [He+10]. To find the particles at the surface the same criterion as for the surface tensions is used, namely if

$$||\mathbf{n}_i|| \geq l \tag{3.44}$$

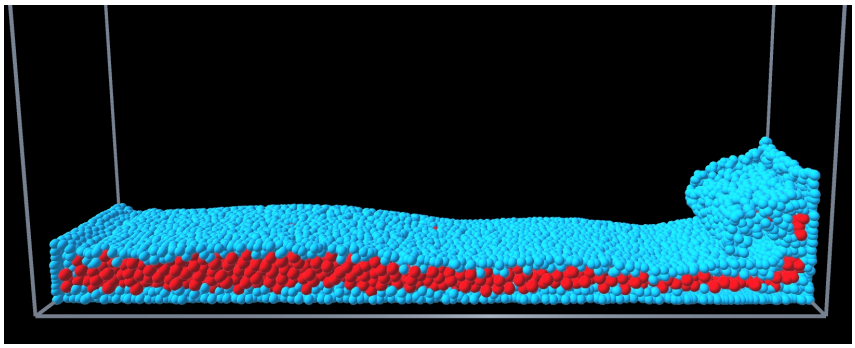


Figure 3.2: The picture shows the an example of how the adaptive framework works, based on the surface of the fluid. The surface and boundary particles (blue) are smaller in size, while the fluid body particles (red) are larger.

where **n** is the inwards surface normal and l is the threshold, then the particles are considered surface particles [Yan+09]. The surface particles then serve as a basis when calculating the distances to the surface and boundaries, denoted ds_i, for each particle i, which are in turn used to compute the smallest distance to the surface for the particle, denoted d_i, where $d_i = min\{ds_{i1}, ds_i i2, \dots\}$. A largest global distance to the surface is also computed, denoted d_{max}. If a particle however is found to be a surface particle, based on criterion (3.44), their smallest distance is set to zero, namely $d_i = 0$.

Secondly, the pressure of a particle are taken into consideration [He+10]. The local pressure is denoted f_i^p while the maximum pressure is denoted f_{max}^p.

Lastly, the level of isolation of a particle from the fluid body is also included as isolated droplets also require a higher level of detail and represent a higher visual complexity [He+10]. The number of neighboring particles for particle i is denoted N_i, while the maximum amount of neighbor over all particles is denoted N_{max}.

Using the parameters above, the definition of the character field function cf_i is as follows:

$$cf_i = \frac{\alpha \frac{d_i}{d_{max}} + \beta(1 - \frac{f_i}{f_{max}}) + \gamma \frac{N_i}{N_{max}}}{\alpha + \beta + \gamma} \tag{3.45}$$

where α, β, and γ are weighting parameters controlling the proportion of the different conditions of the function [He+10], which can be adjusted as desired. The value of these weights, which are presented in section 3.9, have been chosen such

that there are no particles of size $LARGE$ visible on the surface or by any of the boundaries.

Based on the value of the character field function and a threshold value T, it is determined whether a particle should be split into finer particles or combined into one larger particle. Whether particles are split or merged is determined by the following:

$$\begin{cases} \text{SPLIT} & \text{if } cf_i < T \\ \text{MERGE} & \text{if } cf_i \geq 2T \\ \text{REMAIN} & otherwise \end{cases} \tag{3.46}$$

3.6.4 Eye-tracking based Adaptive Framework

The eye-tracking based framework implies that the gaze of a user will dictate the sizes of the particles. Due to the size of a users view point being smaller compared to the whole rendered image [Gue+12], all particles within a radius of the gaze point, henceforth called the gaze radius, are smaller in size, while the particles outside this radius are larger in size, which implies that all particles within the radius r_{gaze} with a center at the view point $\mathbf{r}_{gaze}$ are considered for splitting, if they are not already split. An example is depicted in figure 3.3.

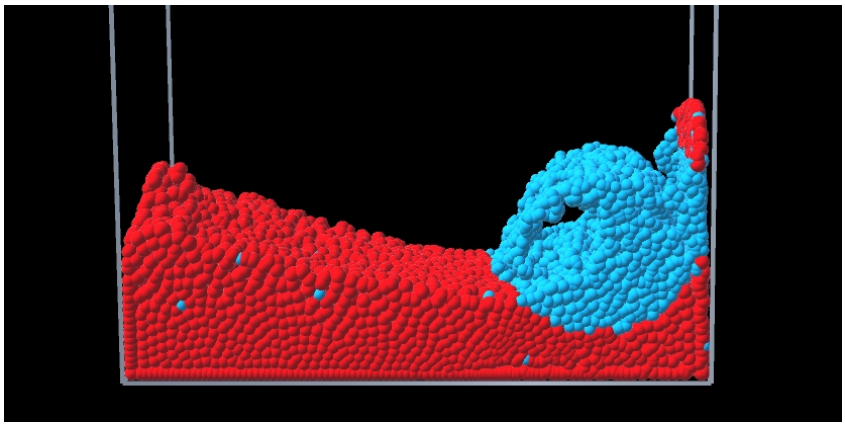


Figure 3.3: The picture shows how the of the adaptive framework based on eye-tracking works based on the gaze point. The particles within the gaze radius (blue) are smaller in size, while the particles outside it (red) are larger.

Using the eye-tracking device Tobii Eye Tracker 4C and the Tobii SDK for Unity [Tob] the user's gaze point is located on the computer screen. In combination with the gaze radius, which is calculated using the focus area of a user and a view percentage of 5% [Gue+12], it is used to dictate the splitting and the merging of the

particles. To measure whether a particle is within the gaze radius, a particle's position is projected onto the screen to find its screen coordinates and then a method similar to identifying collisions with spheres but in two dimensions instead of three, refer to section 3.5, is used where f is defined as

$$f = \text{sgn}\left[||\mathbf{r}_i - \mathbf{r}_{gaze}||^2 - (r_i + r_{gaze})^2\right] \tag{3.47}$$

and whether particles are split or merged is determined by the following:

$$\begin{cases} \text{SPLIT} & \text{if } f < 0 \\ \text{MERGE} & \text{if } f > 0 \\ \text{REMAIN} & otherwise \end{cases} \tag{3.48}$$

3.7 Implementation on the GPU

For the simulation to be interactive it needs to reach an frames per second (FPS) value around 20 and above, which is a very generous minimum considering that virtual reality based applications usually require an frame rate over 60 to prevent nausea. Running the simulation solely on a CPU would not be able to reach the aforementioned requirement when running with few hundred of particles, hence making it an unviable option for a simulation that requires an interactive medium like eye-tracking. All algorithms are therefore partially implemented on the GPU, where there are in total three GPU instances running throughout each frame of the simulation. Each *thread* running on the GPU computes its values for one single particle, and the threads are divided into different *groups* which run in parallell.

The first GPU instance computes the mass-density and pressure term of each particle, as these values are needed to compute the forces. The second instance computes the internal and external forces, proceeds to compute the acceleration and a predicted new position and velocity using time integration, to finally execute the collision handling to correct the predicted values. The third and final instance computes the values needed to decide whether a particle is to be split, merged, or remain unchanged and then proceeds to find these particles.

3.8 Simulation Flow

The simulation starts by initializing all values, such as data structures, constants and pre-computed values. The mass-density and pressure term are computed first in a separate GPU instance directly after the initialization or as the first step in a new

frame. After a synchronization over the CPU a new GPU instance is started where internal and external forces are computed, followed by updating the position and velocity of each particle through time integration, while the collision handling corrects these predicted values. Another synchronization over the CPU takes place before a last GPU instance is started for identifying particles to merge and split, according to the adaptive method used. The merging and splitting then takes place on the CPU, as merging is not possible on the GPU without data race issues. Lastly the particles are rendered and the cycle starts anew. An overview of the flow can be seen in figure 3.4.

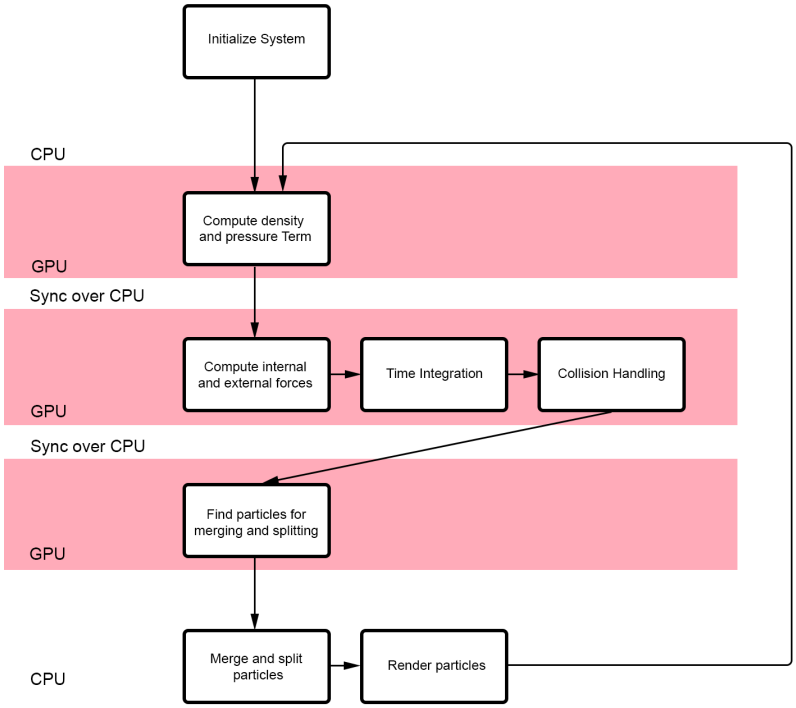


Figure 3.4: This flowchart shows the steps of the simulation, including which parts run on the GPU. Each separate GPU section (red) is its own GPU instance, hence the synchronization over the CPU in-between.

It is important to note that the implementation can be done completely on the GPU, removing the need for synchronizations on the CPU between different steps in the simulation. As Unity is used, this is not a viable option and is also out of scope for this thesis. To read more about implementations of particle based methods on the GPU and CPU, refer to the paper by Dominguez, Crespo, and Gomez-Gesteira [DCG11].

3.9 Parameters

The constant parameters referenced throughout earlier sections, which are used in the simulations are shown in tables 3.1, while the parameters used for the adaptive methods are shown in table 3.2. The parameters that are not present in the tables are all possible to calculate using previously mentioned methods and have been omitted to not cause confusion about which parameters can carelessly be changed.

Description	Symbol	Value
Rest density	ρ_0	998.29
Viscosity	μ	3.5
Surface Tension	σ	0.0728
Gas stiffness	k	3
Time step	Δt	0.01

Table 3.1: The parameters which are used throughout the simulation [Kel06].

Observe that the weight parameters in table 3.2 are used for this specific implementation and evaluation, however they can be modified depending on the scenario, degree of adaptiveness, the number of kernel particles and so on. Yan et al. [Yan+09] and He et al. [He+10] also mention that the weights can be controlled through user input if desired.

Description	Symbol	Value
Geometric weight	α	0.8
Dynamic weight	β	0.01
Isolation weight	γ	0.1
Threshold (3.46)	T	0.225
Number of child particles	n	2
Mass	m_{SMALL}	0.01
Mass	m_{LARGE}	0.02
Kernel particles	x_{SMALL}	50
Kernel particles	x_{LARGE}	20

Table 3.2: The parameters specific for the adaptive methods.

Chapter 4

Evaluation

This sections means to explain how the research question is evaluated as well as which measurements and scenarios are used for the evaluation. To put the results of the two adaptive methods in a perspective, a non-adaptive method is also implemented and measured for each scenario in accordance to what is described in this section. All implementation was done using Unity and by extension HLSL for parallellization.

4.1 Measurements

As the research question is focused on the performance of the eye-tracking based adaptive method compared to the surface based adaptive method, the measurements used need to focus on the performance of the simulation using the different methods. The first measurement to utilize when measuring the performance of a real time interactive simulation is *Frames Per Second* (FPS), which measures how many frames the simulation can produce during 1 second. During one frame, a whole simulation cycle is performed which has been described in section 3.8, and hence the FPS also measure how many cycles the different methods can compute during one second.

While the FPS clearly can show how fast each method produces frames, it does not necessarily alone show which method is more efficient. More particles included in the simulation causes a greater computational need per frame, directly affecting the FPS. As a result the second measurement that is utilized for the evaluation is the *particle count*, or the amount of particles, per frame during the simulation. Observing both the FPS and the particle count give a quantiative characterization of the efficiency of the method.

To further examine potential bottle necks, which are not necessarily visible when only looking at the FPS and particle count, it is important to measure the *execution time* for the different parts of the simulation. Hence the third and the final measurement that is utilized is the execution time of the simulation. The execution time is measured for three different parts of the simulation, the density and pressure computation, which is the first GPU instance in section 3.8, the acceleration computation, time integration and collision handling, which is the second GPU instance, and finally the adaptive step, which includes both the last GPU instance as well as the actual merging and splitting of particles.

4.2 Scenarios

The performance of the adaptive methods is compared in five different scenarios, where each scenario includes a base of 10000 particles. This implies that if all particles are split, there can be a maximum of 20000 particles in the simulation simultaneously. It hence also follows that the regular non-adaptive method always has 20000 particles present in the simulation.

The first scenario is the dambreak scenario, where the bounding box the fluid resides in is expanded allowing sloshing of the fluid. The dambreak scenario is commonly used [Kel06; DCG11; SG11; JKS11; Hua+15; Xu16; Ban+18a] and is hence also used for the evaluation in this thesis. The scenario is simulated for 30 seconds for the FPS and 900 frames for the particle count and execution time. Figure 4.1 shows the water crashing against the wall after the expansion of the bounding box for this scenario.

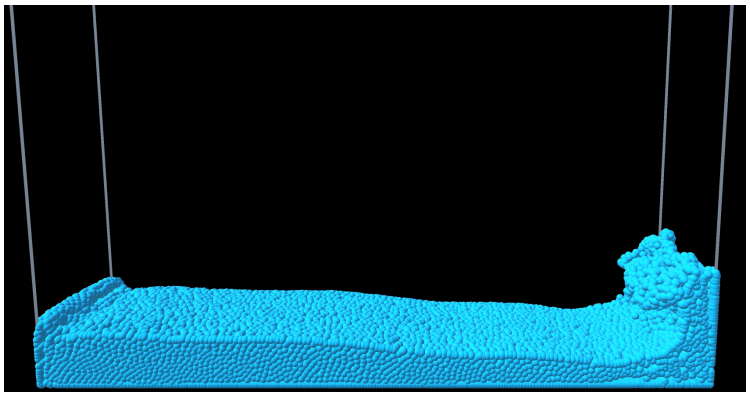


Figure 4.1: Shows how the water crashes up against the wall of the bounding box after the expansion in the dambreak scenario.

The second scenario expands the first, including rigid body objects for the fluid to interact with. This allows the fluid to interact with objects other than the bounding box, showing how the fluid reacts to the collision objects, which forces the collision handling to take into account multiple objects when correcting a predicted particle position. The scenario is simulated for 30 seconds for the FPS and 900 frames for the particle count and execution time. Figure 4.2 shows the scenario layout and how the water is interacting with the obstacles.

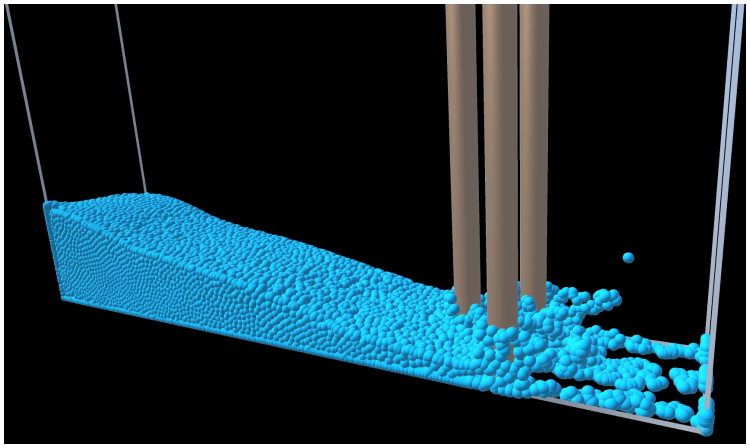


Figure 4.2: Shows how the water interacts with the obstacles after the expansion of the bounding box.

The third scenario drops the fluid in a large bounding box, allowing most of the particles in the fluid to be considered surface particles. This scenario is designed to forcibly try to create as many surface particles as possible using the surface based adaptive method. For more movement to be allowed in the large bounding box, this scenario is measured during 40 seconds for the FPS and 1300 frames for the particle count and execution time. Figure 4.3 shows the large bounding box and how the water is spread out over its area.

The fourth scenario is designed to gather the particles as much as possible in a small bounding box, which would cause most, if not all, particles to be within the gaze radius. It is designed specifically to try to force the eye-tracking based adaptive method to split as many particles as possible and hence having around 20000 particles throughout most of the simulation. The scenario is simulated during 30 seconds for the FPS and 600 frames for the particle time and execution time. Figure 4.4 shows the small bounding box and the gathered particles, which is small enough

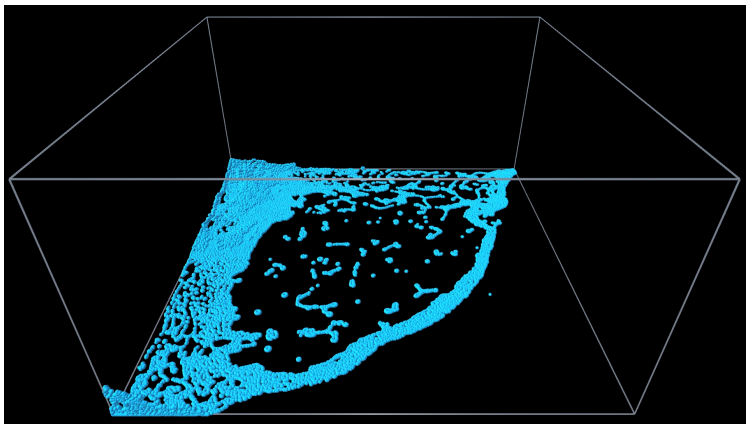


Figure 4.3: Shows how the water spreads out in the large bounding box.

for the lower half of the box to be within the gaze radius.

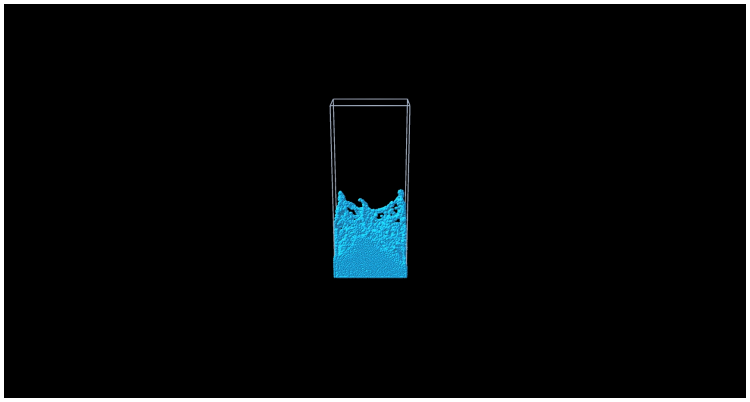


Figure 4.4: Shows how the small bounding box gathers the particles into a small and limited space.

The fifth, and final, scenario is a wave generator whose goal is to continuously create movement in the simulation. A wall of the bounding box is moved back and forth to move the particles, which forces both adaptive methods to continuously have to recalculate which particles that need to be split or merged, while the previous scenarios risk having no continuous merging and splitting once the fluid settles. This scenario is meant to either confirm the results from the previous scenarios or aid in the discussion if the differences are large between the scenarios. Figure 4.5 shows a wave generated by the mentioned moving wall, while it is moving away from the water.

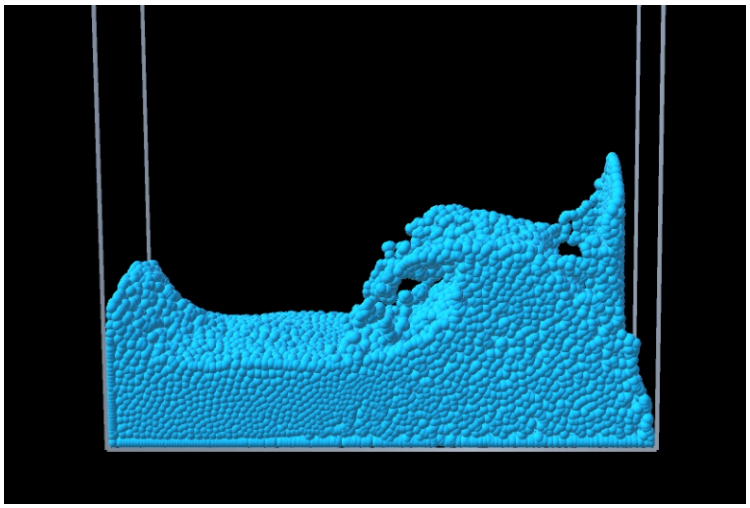


Figure 4.5: Shows a wave generated by the moving wall (right), when it is moving away from the water (towards the right).

4.3 User Study

The eye-tracking based adaptive method is dependant on the tracking of what a user is looking at and hence a user study is conducted to find an average for each of the measurements for the eye-tracking based method. The participants in the study do not need to have any previous experience in eye-tracking or fluid simulations, and are hence not limited by any conditions. Instead it is important to calibrate the eye-tracking device, before starting any of the simulations, for each participant for the device to accurately track the eyes and find the gaze point of the participant. The surface based implementation does not require a user study as it is a numerical simulation implemented so that it does not depend on any user input.

The participants are for each of the five scenarios asked to look wherever they like, as long as they gaze remains somewhere on the water and are informed that the simulation will be restarted if they gaze outside the water. Their gaze on the screen is monitored for the whole duration of each simulation. While this does allow for great variation in where the users are looking, it also allows to more accurately portray the results for the eye-tracking based method.

There are seven participants in this study, all being familiar with computers, but none of them having any previous experience with eye-tracking hardware or fluid simulations. Among the participants there are four who identify as women and three who identify as men, their ages ranging from 10 to 53 years old. Due to the Covid-19

outbreak, all participants are related to the author of the thesis. While it is considered more preferable with a larger group of people to conduct the study, as well as a more varied group, the results of the participants are deemed credible due to the given instructions, where the author conducts the calibration of the hardware.

4.4 Hardware

The following hardware was used for the computations

- GPU: GeForce RTX 2070
- CPU: Intel Core i5-9400F CPU @ 2.90GHz
- RAM: Corsair 16 GB DDR4
- Eye-tracking: Tobii Eye Tracker 4C

Chapter 5

Results

This section presents the results obtained for five different scenarios, where each scenario is presented in an individual section. There are seven participants in the user study conducted for the eye-tracking based adaptive method, whose results are represented by dotted black lines in the figures in this section. Figures that show the visualization of the simulation flow for both methods in each scenario can be found in appendix A.

5.1 Dambreak

The dambreak scenario involves a so called dam breaking, releasing the water into a larger container which then causes waves and sloshing when the water crashes into the wall of the expanded bounding box. Figure 5.1 shows the FPS during the scenario, measured for 30 seconds. It can be observed that the FPS for the eye-tracking based adaptive method, black in the figure, is significantly higher than the surface based adaptive method, both for the average result and also for each individual participant's result (dotted black lines). The surface based method however seems to have an FPS similar to the one of the regular method.

In figure 5.2 the evolution of the amount of particles present in the simulation is shown. It can be seen that the amount of particles for all results relating to the eye-tracking based adaptive method is significantly lower than for the surface based method. The surface based method however does have a lower amount of particles than the regular non-adaptive method.

Table 5.1 shows the execution time in milliseconds for different parts of the algorithm. It can be observed that both adaptive methods are significantly faster than the

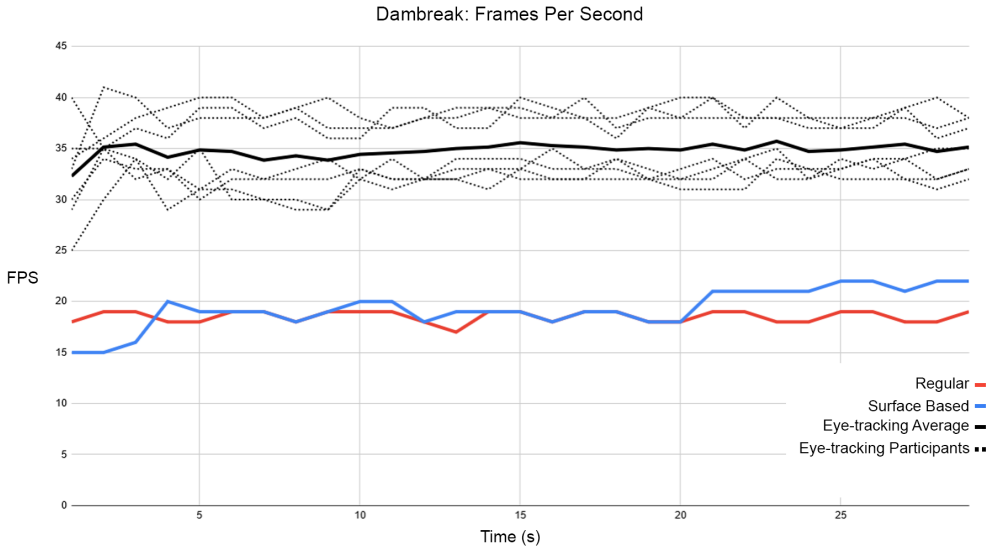


Figure 5.1: The frames per second (FPS) measured over 30 seconds for the dambreak scenario.

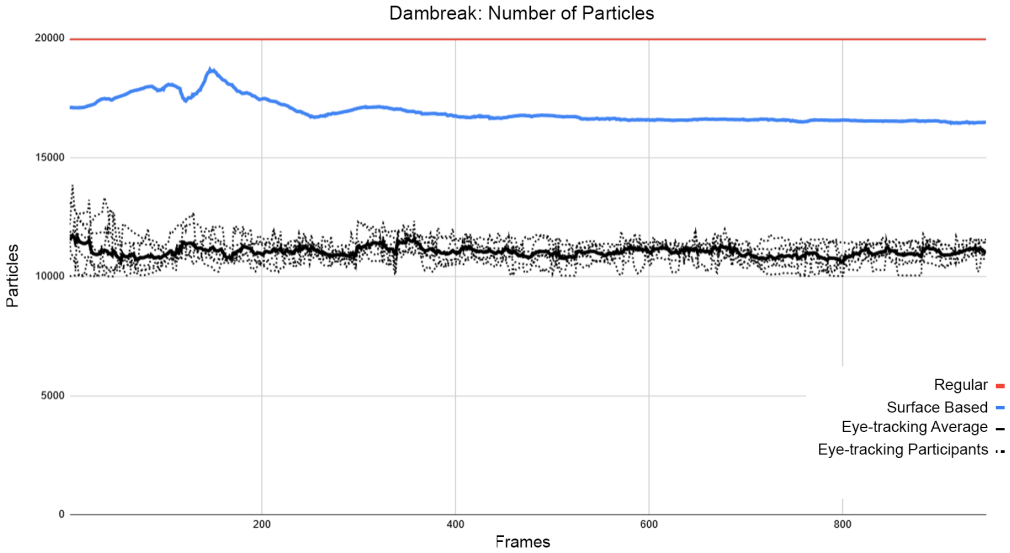


Figure 5.2: The amount of particles measured over 900 frames for the dambreak scenario.

regular method in terms of the computation of Density and Pressure and the computation of Acceleration, Integration and Collision, albeit the eye-tracking based

method seems to be faster than the surface based method. However, the surface based method cannot be said to be faster than the regular method in total execution time, which is also shown in the table. The time required for the adaptive step, which the regular non-adaptive method lacks, in combination with the rest causes the surface based method to require as much time for execution as the regular method, while the eye-tracking based method requires significantly lower time in total.

Computation	Regular	Surface Based	Eye-tracking Average
Density & Pressure	10.42	7.263	4.398 ± 0.229
Acceleration, Integration & Collision	13.43	9.499	5.544 ± 0.321
Adaptiveness	-	8.219	4.265 ± 0.322
Total	23.84	24.98	14.21 ± 0.813

Table 5.1: The table presents how much time in milliseconds the different methods require to compute the different parts of the simulation for the Dambreak scenario.

5.2 Dambreak with Collision

This scenario is similar to the dambreak scenario, with an addition of rigid body obstacles that the water collides against. Figure 5.3 shows the FPS for the scenario measured during 30 seconds, where it can be seen that the results for each participant in the user study for the eye-tracking based method are higher than both the surface based method and the regular non-adaptive method. While the surface based adaptive method seems to have an FPS higher than the regular method, the difference is quite small. Figure 5.4 does however show that the amount of particles in the surface based adaptive method is significantly lower than the regular method, while the eye-tracking based method has the lowest count of particles per frame throughout the whole simulation.

In table 5.2 it can once again be seen that the surface based and eye-tracking based methods both have a significantly shorter execution time than the regular non-adaptive method for the common parts of the simulation. However, the total time required for the execution of the simulation is similar between the surface based adaptive method and the regular non-adaptive method, due to the time the adaptive step requires. The

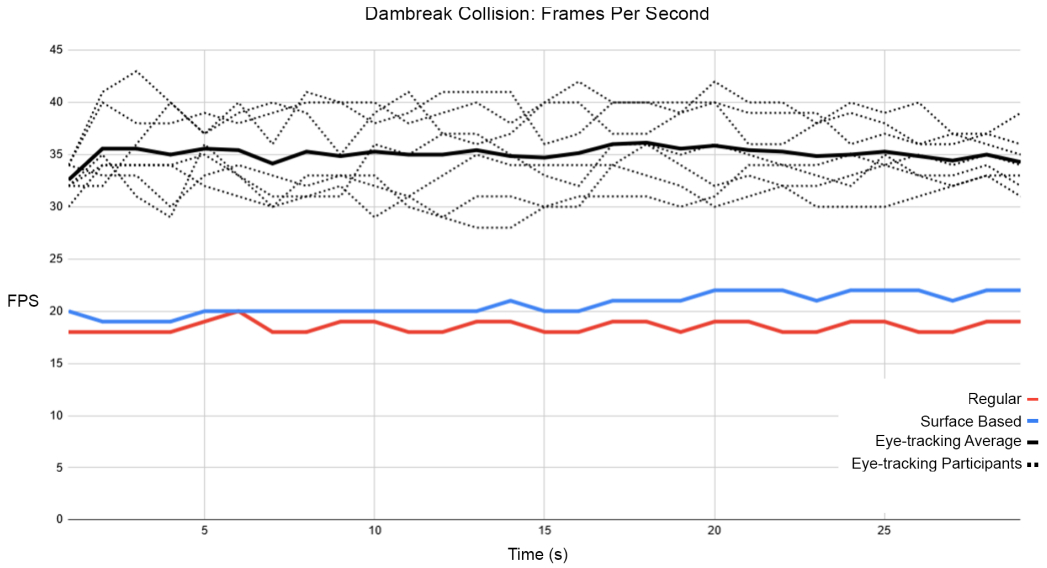


Figure 5.3: The frames per second (FPS) measured over 30 seconds for the dambreak scenario with collision.

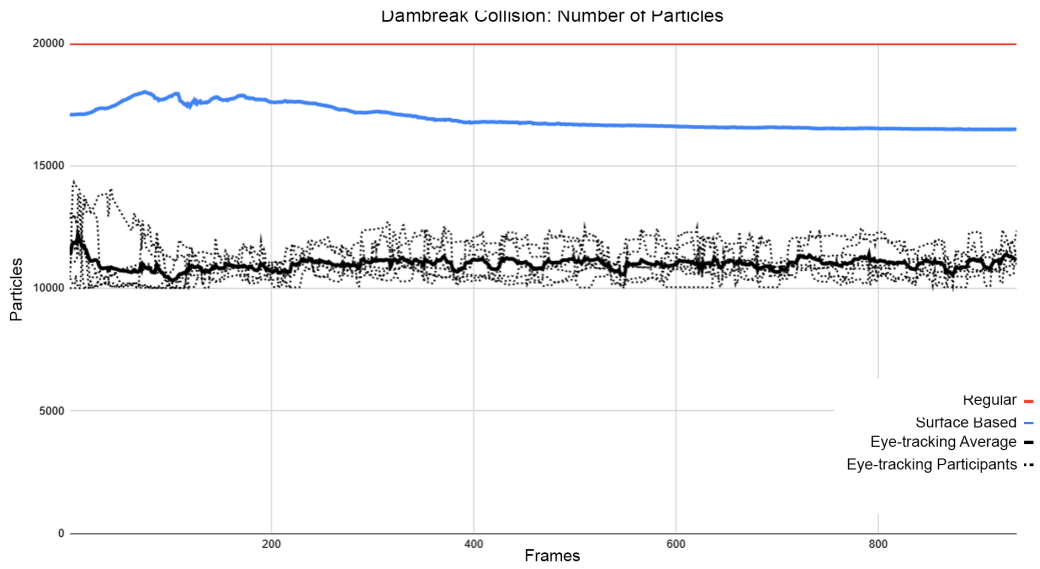


Figure 5.4: The amount of particles measured over 900 frames for the dambreak scenario with collision.

eye-tracking based method instead has a shorter execution time compared to the surface based method in all steps of the simulation, resulting in a faster simulation.

Computation	Regular	Surface Based	Eye-tracking Average
Density & Pressure	9.970	6.616	4.289 ± 0.283
Acceleration, Integration & Collision	13.63	9.109	5.713 ± 0.482
Adaptiveness	-	7.844	4.472 ± 0.467
Total	23.60	23.57	14.47 ± 1.221

Table 5.2: The table presents how much time in milliseconds the different methods require to compute the different parts of the simulation for the dambreak scenario with collision.

5.3 Large Bounding Box

Using a large bounding box allows the particles to spread out, creating more potential surface particles. In table 5.3 it can be observed that while the surface based method has a significantly lower execution time for the parts of the simulation that are also used in the regular method, its total is similar to the total of the regular method. The eye-tracking based method instead has a significantly faster execution time for each part of the simulation and in total.

Computation	Regular	Surface Based	Eye-tracking Average
Density & Pressure	10.02	6.714	4.748 ± 0.429
Acceleration, Integration & Collision	13.31	9.206	6.060 ± 0.557
Adaptiveness	-	8.341	4.500 ± 0.456
Total	23.84	24.26	15.31 ± 1.403

Table 5.3: The table presents how much time in milliseconds the different methods require to compute the different parts of the simulation using a large bounding box.

Figure 5.5 shows the FPS for this scenario measured during 40 seconds, where it

can be observed that the FPS for the surface based method is close to identical to the FPS of the regular non-adaptive method. The eye-tracking based method instead has a significantly higher FPS, albeit the results in the beginning of the simulation is lower than the rest of the simulation

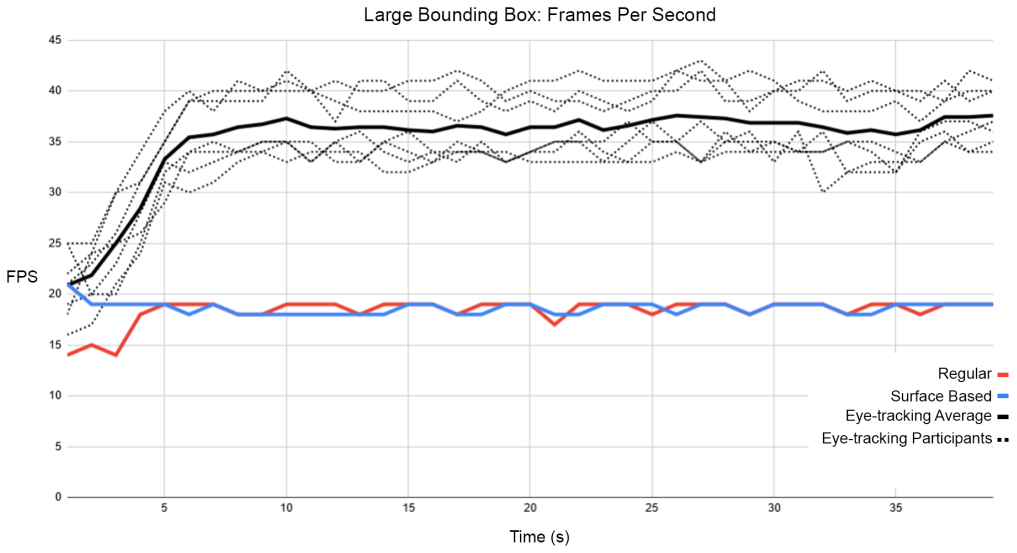


Figure 5.5: The frames per second (FPS) measured over 40 seconds using a large bounding box which spreads the particles out.

As the particles spread out over the larger area they can cover, it can in figure 5.6 be seen that the particle count falls rapidly after the first few hundred frames for the eye-tracking based method. The surface based method does have a lower amount of particles compared to the regular method, however it has a significantly larger amount of particles compared to the eye-tracking based method.

5.4 Small Bounding Box

Using a small bounding box allows for the particles to gather at a limited space, which causes most, if not all, particles to be within the gaze radius for the eye-tracking based adaptive method. In figure 5.7 it can be seen that the amount of particles varies greatly for the eye-tracking based method with no apparent advantage to the surface based method. Instead the eye-tracking based method has a higher particle count, in average, after around 100 frames into the simulation compared to the surface based method, which has a stabilized particle count after around 200 frames

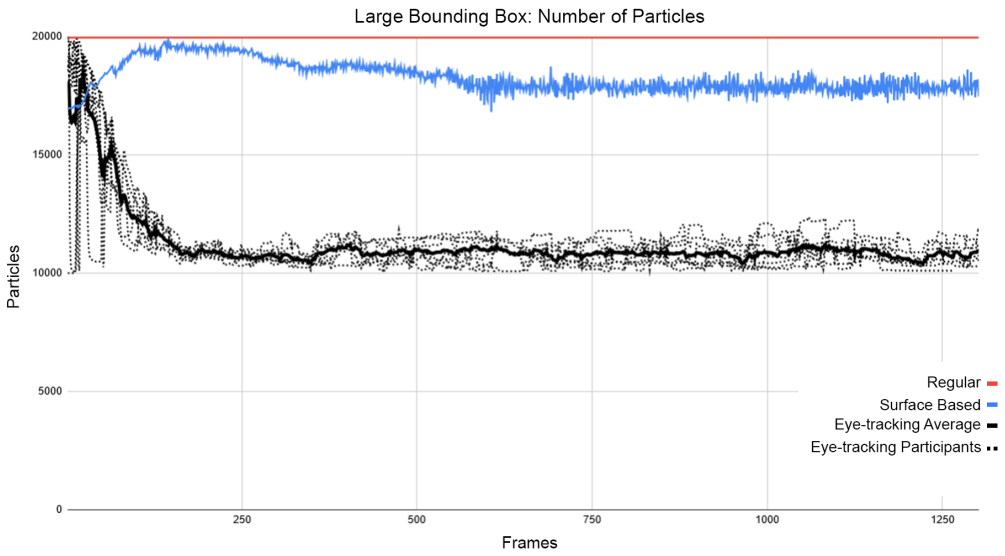


Figure 5.6: The amount of particles during the scenario measured for 1300 frames, using a large bounding box which spreads the particles out.

into the simulation, while having a significantly lower particle count compared to the regular non-adaptive method. At certain times throughout the frames of the simulation, for both individual cases and for the average, the eye-tracking based method has an amount of particles close to the amount of regular method, while the amount of particles for the surface based method constantly stays significantly lower, except for in the very beginning of the simulation.

As seen in figure 5.8, the eye-tracking based method does not have a significant advantage compared to the surface based method, instead having two cases which constantly have a lower FPS compared to the surface based method, which is more clearly shown in figure 5.9. The figure also shows that both cases seem to perform similarly to the regular non-adaptive method. The average result as well as the rest of the individual cases for the eye-tracking based method seem to result in similar FPS to the surface based method.

In table 5.4 it can be seen that, unlike in previous scenarios, the eye-tracking based method has no significant advantage in execution time compared to both the surface based method and the regular non-adaptive method. Instead it has a similar result to the one of the surface based method, albeit a bit higher. It can however also be seen that the standard deviation for the results for the eye-tracking based method is high.

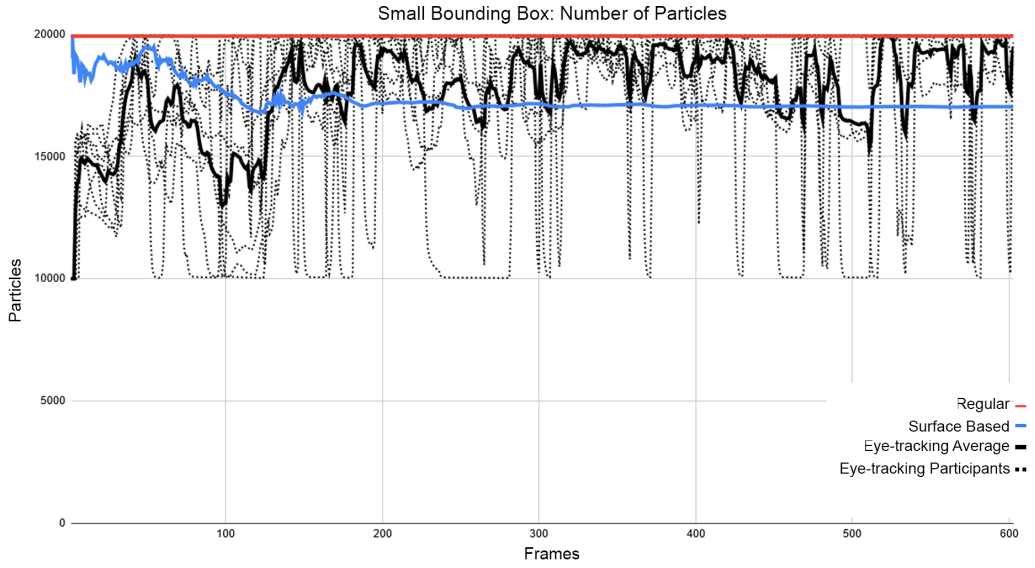


Figure 5.7: The amount of particles measured for 600 frames, using a small bounding box which gathers the particles.

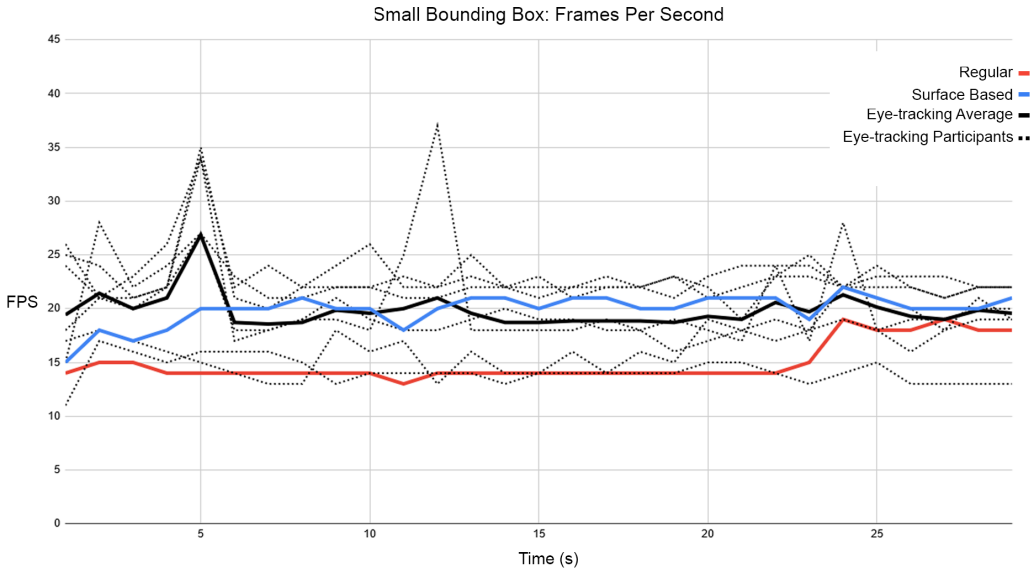


Figure 5.8: The frames per second (FPS) measured over 30 seconds, using a small bounding box which gathers the particles.

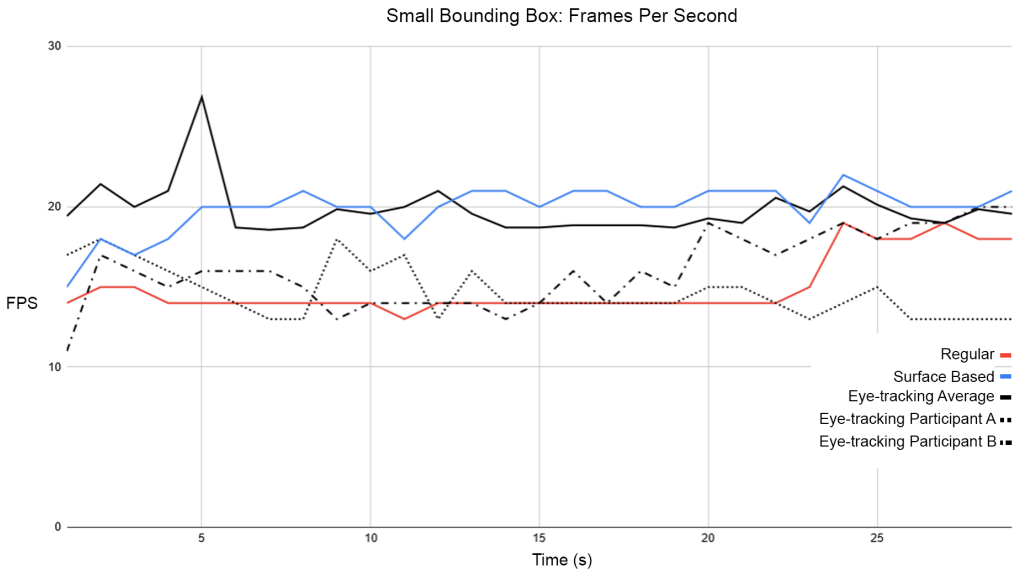


Figure 5.9: The FPS measured over 30 seconds comparing the result of two participants using the eye-tracking based method to the result of the average FPS for the eye-tracking based method, the surface based method, and the regular non-adaptive method.

Computation	Regular	Surface Based	Eye-tracking Average
Density & Pressure	11.63	6.372	7.136 ± 1.180
Acceleration, Integration & Collision	15.27	8.761	9.792 ± 1.724
Adaptiveness	-	7.541	6.525 ± 1.627
Total	26.90	22.67	23.45 ± 4.459

Table 5.4: The table presents how much time in milliseconds the different methods require to compute the different parts of the simulation using a small bounding box.

5.5 Wave Generator

By creating continuous movement, this scenario forces both adaptive methods to constantly split and merge particles. It can be seen in figure 5.10 that the FPS, while varying for the different participants, is significantly higher for the eye-tracking based method. The surface based method instead has a result similar to the one of

the regular non-adaptive method.

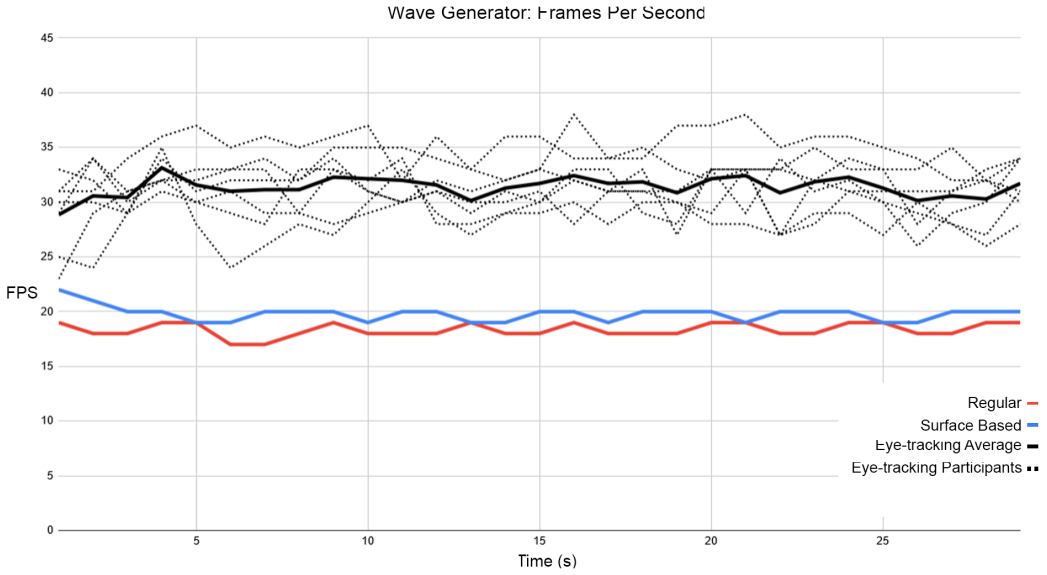


Figure 5.10: The graph shows the frames per second (FPS) measured over 30 seconds, where movement is constantly created during the simulation.

Similarly, the surface based method has total execution time similar to the one of the regular method, even though it has a faster execution time for the parts of the simulation the methods have in common, which can be seen in table 5.5. The eye-tracking based method has a significantly faster execution time in all individual parts of the simulation and a faster execution time in total.

Computation	**Regular**	**Surface Based**	**Eye-tracking Average**
Density & Pressure	9.815	6.991	4.838 ± 0.295
Acceleration, Integration & Collision	12.75	9.260	6.199 ± 0.410
Adaptiveness	-	8.261	4.694 ± 0.483
Total	22.57	24.51	15.73 ± 1.179

Table 5.5: The table presents how much time in milliseconds the different methods require to compute the different parts of the simulation when generating continuous movement in the simulation.

It can be seen in figure 5.11 that the eye-tracking based method has a lower, albeit varying, amount of particles throughout the simulation, compared to the surface based method. While the surface based method has a lower amount of particles compared to the regular non-adaptive method, it is constantly higher than the eye-tracking based method. Unlike the other scenarios, the particle count for the surface based method is varying to a larger degree.

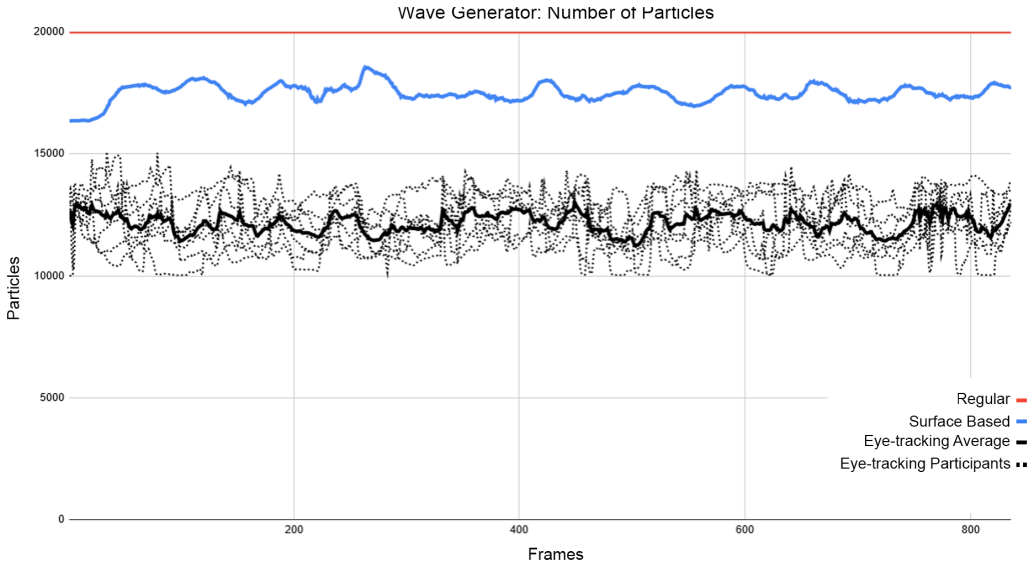


Figure 5.11: The amount of particles measured for 850 frames, while constantly creating movement.

Chapter 6

Discussion and Future Work

This section aims to discuss the results in correlation to the research question and the improvements that can be made to the implementation to further improve the quality of the results. Further, potential future work will be mentioned and discussed.

6.1 Performance

The results quite clearly show that the eye-tracking based adaptive method has a better performance in all scenarios except one, which is the scenario using a small bounding box. This scenario was in particular designed to force the eye-tracking based method to have a lot of split particles, hopefully forcing all particles to be split and resulting in having 20000 particles simultaneously throughout the simulation. While it did have a lower particle count during certain parts of the simulation as seen in figure 5.7, it can be said that having all particles within the gaze radius certainly did affect the method for the worse, both lowering its resulting FPS and increasing the execution time. It can be seen in figure A.7 that the varying particle count is a result of the participants focusing on different parts of the fluid, even though the particles were mostly gathered, however it is also clear that most particles were inside the gaze radius.

For the remaining four scenarios, the eye-tracking based adaptive method has the best results with lower particle counts, a higher FPS and a lower execution time. Except for the fact that the total amount of particles is lower in these scenarios compared to the surface based method, a potential explanation for the lower FPS is how many particles are in need of splitting and merging. For the surface based method, particles are split if they are considered close enough to the surface or to a boundary, creating a larger area that can potentially be surface or boundary particles. For

the eye-tracking based method, only particles within the gaze radius are considered for splitting while the rest should remain the same if they are not merged. As the gaze radius is based on the field of view and the focus area of the eye [Gue+12], the amount of particles that fit within this radius will vary depending on the simulation. As the simulation comes closer to the camera, the amount of particles fitting within the gaze radius will decrease. Of course, if the eye-tracking based method was to be used for a simulation far away from the camera, the amount of particles needed in the simulation would increase, as showed in the results for the scenario using a small bounding box in section 5.4.

The lower particle count for the eye-tracking based method has most probably contributed the most to the higher FPS and the faster execution time, as fewer particles need to be handled throughout the simulation. However, as stated, the method lacks when most of the particles are within the gaze radius. It can hence be argued that the method is more suitable when simulating either larger bodies of water which cannot fit within the gaze radius or when the water is within a closer vicinity of the camera. Improvements addressing this issue is discussed further in section 6.4. The scenarios used in this thesis can also be critiqued, as differently designed scenarios might give other types of results, depending on their design.

The surface based adaptive method had a similar performance throughout all five scenarios for all three measurements, where both the FPS and the execution time were similar to the regular non-adaptive method. While it in many cases performed on par with the regular non-adaptive method, it is not to say that the method is ineffective. Instead the implementation used for this thesis could be to blame, as both Yan et al. [Yan+09] and He et al. [He+10] successfully have used this method to improve the non-adaptive SPH performance wise.

6.2 Implementation

As previously discussed, the surface based adaptive method was shown to have results similar to the regular non-adaptive method, even though it had a lower particle count throughout the simulations. One explanation for this can be that the computation for the adaptive step of the simulation step was too ineffective, as suggested by the execution time results. As the method has a faster execution time compared to the regular method in regards to the parts they had in common, it is by no means as slow as the regular method for those specific parts. The issue seems to rather lie in the total execution time which is similar for both methods, the difference being the adaptive step which is included in the surface based method.

As mentioned in section 3.8 the actual merging and splitting takes place on the CPU, which could provide an explanation to the time required to perform the adaptive step. It would of course have been better to perform these computations on the GPU instead as it would have increased performance, however Unity and HLSL did not allow for the necessary implementation. The most prominent issue with specifically merging on the GPU are the potential data race issues that can arise. As there needs to be two particles for merging and which two particles are merged together is not necessarily possible to decide deterministically using HLSL, multiple particles can potentially choose the same particle to merge with.

For a single particle to claim another particle to merge with, the threads in the GPU need to communicate this information to each other. However, Unity in combination with HLSL does not allow this as the threads run in so called *groups* and two threads that belong to different groups have no means of communication. As each particle is considered a thread, that single particle cannot guarantee to be able to reach out to all essential threads with the necessary information to avoid race issues. It is important to note one could possibly change how particles are split among threads to make sure all existing threads are instead within the same group and hence possibly enable more communication. However, there is no guarantee that there would be enough support in Unity nor HLSL for using a mutex or locks to avoid data race issues, as the means to parallellize in Unity is mainly built for rendering where little to no communication between groups is necessary.

Another potential problem with this implementation is the fact that it is partially implemented on the CPU and partially on the GPU, as seen in section 3.8. As shown by Dominguez, Crespo, and Gomez-Gesteira [DCG11], an implementation fully on the GPU has a faster execution time for the simulation, compared to a partial implementation. As previously mentioned, Unity and HLSL does not allow communication between groups and there is hence no possibility to synchronize the data between each GPU instance depicted in figure 3.4. If the distribution of threads were to be implemented differently so that they all reside within a single group, it might be possible to synchronize the data partially, as the adaptive step would still need to be on the CPU as previously mentioned. Suggestions to get around this problem is further discussed in section 6.4, however worth noting is that it was decided that a full GPU implementation was decided to be outside the scope of this thesis.

6.3 General Discussion

While the results seem to indicate that an eye-tracking based adaptive implementation has potential to increase the performance of the simulation, one might question if this method is needed to begin with. The scenario using a small bounding box indicates that water bodies further away might impose a problem for the method, however it could be argued that a water body further away might not need a more detailed simulation or a simulation with a higher resolution, which adaptive methods generally try to achieve. Similarly there is no need for greater detail or resolution for the thing one cannot see, which is achieved by both the surface based adaptive method, as you cannot see the larger particles underneath the surface, and the eye-tracking based adaptive method even if different ways. However, unlike the surface based method, the eye-tracking based method takes this a step further and includes what an eye can perceive, further decreasing the computation needed, as suggested by the results, possibly making it a viable option for real-time applications that desire more realistic water simulations.

This thesis does not focus on sustainability, as it is mainly focuses on water simulations and how to make them more performance effective, suggesting an eye-tracking based method. It is believed that this method will mainly be used in future Virtual Reality environments, where eye-tracking is becoming increasingly common, to limit the simulations and reach acceptable performance for these environments by focusing the computational power where the user is looking. Worth noting however, is that the results suggest a decrease in required computational power overall when using the eye-tracking based adaptive method in the correct context, which can be considered beneficial for future sustainable development of the Virtual Reality environments that are most likely to use this method, due to the decreased computational power needed when computing the simulations.

A greater downside to the eye-tracking based method is the fact that it requires special hardware, an eye-tracker, to function. While it does seem to become increasingly common for Virtual Reality headsets to include eye-tracking, the hardware is not widely used generally. As the industry for Virtual Reality and eye-tracking expands, this method can become more relevant as the hardware most probably then will be more common, however until then this method will be limited by the requirement for hardware.

6.4 Future Work

As has been mentioned, one big limitation with the implementation for this thesis is the fact that it is partially implemented on the CPU and partially on the GPU, which most probably has affected the results, especially for the surface based adaptive method. To hence confirm the results of this thesis it would be beneficial to replicate it, however change either the distribution of particles to the threads aiming to have only one group of threads or instead of using Unity in combination with HLSL using a language that would allow for more control over the threads, such as C, which also has support for using a mutex and locks. This would however require either implementing a rendering system or finding a package which renders the simulation.

This thesis has not focused on the realism of the methods, as it has simply been assumed. It could however be important to also further examine the realism of the suggested eye-tracking based adaptive method or an improved variant to other adaptive methods that are used, as this can confirm the usability and perceived realism of the method. A study of perceived realism for the eye-tracking based method would also suggest how useful this method is in different types of environments, such as games or for larger more scientifically oriented simulations to mention a few. Worth noting is however that this type of study should be done including a surface reconstruction algorithm for more water like visuals, rather than the spheres that are shown in this thesis.

One suggestion that could potentially change the impression of the realism of the simulation as well as the performance of the eye-tracking based method, is to use multiple sizes to create a smoother transition between the smaller particles and the larger ones. This thesis utilizes two different particle sizes for simplicity, however it also affects the stability in a sense. If a particle is split in more child particles, the distribution of these child particles is important as the local pressure could cause instabilities for the whole simulation. While this thesis does suggest how to handle child particles for an arbitrary number, numbers other than two have not been tested and neither has multiple sizes for the simulation. However, as stated, it remains an interesting suggestion to test and further examine due to the effect it can have on realism and performance.

In this thesis all particles are projected to the screen when deciding whether a particle is within the gaze radius or not, which implies that particles that are underneath the surface are also split or merged. To further improve the eye-tracking based adaptive

method, one could either limit how far underneath the immediate visible layer particles should be split or one could combine the eye-tracking based method with the surface based method. Combining the two methods could further lower the amount of particles needed for the simulation, most probably allowing an even better performance, as a smaller area of particles are considered for splitting. It would however most probably also be of great importance to measure the realism of the suggested combination and also measure if the accuracy of the simulation is affected as there will be a very limited amount of particles that are split and merged, which could imply that there is no greater difference in detail and accuracy from running a non-adaptive method with a lower resolution, which in turn would render the method unnecessary.

If the implementation is improved upon, by for example making an implementation fully on the GPU, which makes it possible to reach a higher FPS and a faster execution time than it has been possible to reach in this thesis, it could be worth looking into having a lower time step for a more accurate simulation. As mentioned in section 3.3, the pressure term is computed using a modified version of the ideal gas state equation, which can be very unstable. There are however multiple other suggestions on how to calculate the pressure term that can be considered more stable than the one used for this thesis, many of which require smaller time steps. Hence improving this implementation for a better performance could allow for smaller time steps without affecting the perceived performance, allowing the use of a potentially more stable pressure term computation. In the case of an more accurate calculation overall, it can also be argued that the need for finer detail might be more relevant where a user is watching as opposed to somewhere else in the simulation, making it more relevant for an eye-tracking based method. This is however only speculation.

Chapter 7

Conclusions

In four out of five scenarios the eye-tracking based adaptive method performs significantly better than the surface based adaptive method. However the one scenario where it performs similarly or worse than the surface based method indicates possible problems performance wise for the eye-tracking based method as previously discussed. Nonetheless, to clearly answer the research question of how effective an eye-tracking based adaptive algorithm is compared to a numerical counterpart, which is the surface based method, when creating a water simulation using SPH, then the eye-tracking based adaptive method is significantly more efficient than the surface based method when the simulation is close enough to the camera so that all particles are not within the gaze radius.

As it has been discussed, the results can change depending on which type of scenarios are used but also if the implementation is instead implemented for example fully on the GPU or using another language and programming environment. However it can also be concluded that using an eye-tracking based adaptive method can be of interest, especially when eye-tracking hardware becomes more accessible as well as if it is possible to use more stable pressure calculations with smaller time steps without negatively impacting the performance. It is especially interesting to further examine the use of eye-tracking within interactive or real-time fluid simulations for visual purposes and perceived realism, similarly to how level of detail is used for polygons within computer games etc, as what the user cannot perceive usually does not too be as exact.

Acknowledgements

I would firstly like to thank Johan Hoffman for the continuous support and help during the thesis and for giving me the opportunity to realize this idea to begin with. I would also like to thank the participants of the user study for the eye-tracking based method and the Visualization Studio at KTH for their help with this thesis. Lastly, I would also like to thank my family and the friends who have been helping me to continuously stay motivated and who have been providing me with both wise and encouraging words while working on this thesis and otherwise during my five years at KTH.

Bibliography

[Ada+07] Bart Adams et al. “Adaptively sampled particle fluids”. In: *ACM SIGGRAPH 2007 papers*. 2007, 48–es.

[ATW13] Ryoichi Ando, Nils Thürey, and Chris Wojtan. “Highly adaptive liquid simulations on tetrahedral meshes”. eng. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), pp. 1–10. ISSN: 07300301.

[Ban+18a] Xiaojuan Ban et al. “Adaptively stepped SPH for fluid animation based on asynchronous time integration”. In: *Neural Computing and Applications* 29.1 (2018), pp. 33–42.

[Ban+18b] Stefan Band et al. “Pressure boundaries for implicit incompressible SPH”. In: *ACM Transactions on Graphics (TOG)* 37.2 (2018), pp. 1–11.

[Bar06] Brian Arthur Barran. “View dependent fluid dynamics”. PhD thesis. Texas A&M University, 2006.

[BXH10] Christopher Batty, Stefan Xenos, and Ben Houston. “Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids”. In: *Computer Graphics Forum*. Vol. 29. 2. Wiley Online Library. 2010, pp. 695–704.

[BT07] Markus Becker and Matthias Teschner. “Weakly compressible SPH for free surface flows”. In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association. 2007, pp. 209–217.

[Bot+10] Elkin Botia-Vera et al. “Three SPH novel benchmark test cases for free surface flows”. In: *5th ERCOFTAC SPHERIC workshop on SPH applications*. 2010, pp. 146–153.

[BK11] Rinchai Bunlutangtum and Pizzanu Kanongchaiyos. “Enhanced view-dependent adaptive grid refinement for animating fluids”. In: *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*. 2011, pp. 415–418.

[Del+09] L Delorme et al. “A set of canonical problems in sloshing, Part I: Pressure field in forced roll—comparison between experimental results and SPH”. In: *Ocean Engineering* 36.2 (2009), pp. 168–178.

[DC99] Mathieu Desbrun and Marie-Paule Cani. “Space-time adaptive simulation of highly deformable substances”. In: (1999).

[DG96] Mathieu Desbrun and Marie-Paule Gascuel. “Smoothed particles: A new paradigm for animating highly deformable bodies”. In: *Computer Animation and Simulation’96*. Springer, 1996, pp. 61–76.

[DCG11] Jose M Dominguez, Alejandro JC Crespo, and Moncho Gomez-Gesteira. “Optimization strategies for parallel CPU and GPU implementations of a meshfree particle method”. In: *arXiv preprint arXiv:1110.3711* (2011).

[FM96] Nick Foster and Dimitri Metaxas. “Realistic animation of liquids”. In: *Graphical models and image processing* 58.5 (1996), pp. 471–483.

[GM82] RA Gingold and JJ Monaghan. “Kernel estimates as a basis for general particle methods in hydrodynamics”. In: *Journal of Computational Physics* 46.3 (1982), pp. 429–453.

[Gis+19] Christoph Gissler et al. “Interlinked SPH pressure solvers for strong fluid-rigid coupling”. In: *ACM Transactions on Graphics (TOG)* 38.1 (2019), pp. 1–13.

[Gre10] Simon Green. “Particle simulation using cuda”. In: *NVIDIA whitepaper* 6 (2010), pp. 121–128.

[Gue+12] Brian Guenter et al. “Foveated 3D graphics”. In: *ACM Transactions on Graphics (TOG)* 31.6 (2012), pp. 1–10.

[HW65] Francis H Harlow and J Eddie Welch. “Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface”. In: *The physics of fluids* 8.12 (1965), pp. 2182–2189.

[He+10] Jian He et al. “Real-time adaptive fluid simulation with complex boundaries”. In: *The Visual Computer* 26.4 (2010), pp. 243–252.

[HHK08] Woosuck Hong, Donald H House, and John Keyser. “Adaptive particles for incompressible fluid simulation”. In: *The Visual Computer* 24.7-9 (2008), pp. 535–543.

[Hop97] Hugues Hoppe. “View-dependent refinement of progressive meshes”. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 1997, pp. 189–198.

[Hua+15] Chen Huang et al. “Parallel-optimizing SPH fluid simulation for realistic VR environments”. In: *Computer Animation and Virtual Worlds* 26.1 (2015), pp. 43–54.

[JLL09] Hai-liang Jin, Xiao-ping Lu, and Hui-jie Liu. “View-dependent fast real-time generating algorithm for large-scale terrain”. In: *Procedia Earth and Planetary Science* 1.1 (2009), pp. 1147–1151.

[JKS11] Eunchan Jo, Doyub Kim, and Oh-young Song. “A new SPH fluid simulation method using ellipsoidal kernels”. In: *Journal of visualization* 14.4 (2011), pp. 371–379.

[KS12] Masayuki Kawai and Yohei Suzuki. “Adaptive Parameter Adjustment of a Real-Time SPH Simulation for Interactive Virtual Environments: Application to Parallel Processing”. In: *2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. IEEE. 2012, pp. 692–697.

[Kel06] Micky Kelager. “Lagrangian fluid dynamics using smoothed particle hydrodynamics”. In: *University of Copenhagen: Department of Computer Science* (2006).

[KIC06] Janghee Kim, Insung Ihm, and Deukhyun Cha. “View-Dependent Adaptive Animation of Liquids”. In: *ETRI journal* 28.6 (2006), pp. 697–708.

[KNO14] Woojong Koh, Rahul Narain, and James F O’Brien. “View-Dependent Adaptive Cloth Simulation.” In: *Symposium on Computer Animation*. 2014, pp. 159–166.

[KB17] Dan Koschier and Jan Bender. “Density maps for improved SPH boundary handling”. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2017, pp. 1–10.

[Lau13] Jens Christian Morten Laursen. “Improving the Realism of Real-Time Simulation of Fluids in Computer Games”. In: (2013).

[LGF04] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. “Simulating water and smoke with an octree data structure”. In: *ACM transactions on graphics (TOG)*. Vol. 23. 3. ACM. 2004, pp. 457–462.

[Luc77] Leon B Lucy. “A numerical approach to the testing of the fission hypothesis”. In: *The astronomical journal* 82 (1977), pp. 1013–1024.

[LE97] David Luebke and Carl Erikson. “View-dependent simplification of arbitrary polygonal environments”. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 1997, pp. 199–208.

[MM13] Miles Macklin and Matthias Müller. “Position based fluids”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), pp. 1–12.

[Mon12] Joseph J Monaghan. “Smoothed particle hydrodynamics and its diverse applications”. In: *Annual Review of Fluid Mechanics* 44 (2012), pp. 323–346.

[MCG03] Matthias Müller, David Charypar, and Markus Gross. “Particle-based fluid simulation for interactive applications”. In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association. 2003, pp. 154–159.

[Poh+16] Daniel Pohl et al. “Concept for using eye tracking in a head-mounted display to adapt rendering to the user’s current visual field”. In: *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*. 2016, pp. 323–324.

[Raj11] Perseedoss Rajiv. “Lagrangian Liquid Simulation using SPH”. In: (2011).

[SG11] Barbara Solenthaler and Markus Gross. “Two-scale particle simulation”. In: *ACM SIGGRAPH 2011 papers*. 2011, pp. 1–8.

[SP09] Barbara Solenthaler and Renato Pajarola. “Predictive-corrective incompressible SPH”. In: *ACM SIGGRAPH 2009 papers*. 2009, pp. 1–6.

[Sta99] Jos Stam. “Stable fluids”. In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 1999, pp. 121–128.

[SF95] Jos Stam and Eugene Fiume. “Depicting fire and other gaseous phenomena using diffusion processes”. In: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. 1995, pp. 129–136.

[Tob] Tobii. *Tobii Unity SDK for Desktop (Public documentation)*. `https://github.com/tobiitech/unity-sdk`. Accessed last: 2020-05-15.

[UHT17] Kiwon Um, Xiangyu Hu, and Nils Thuerey. “Perceptual evaluation of liquid simulation methods”. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–12.

[Wei+16] Martin Weier et al. “Foveated real-time ray tracing for head-mounted displays”. In: *Computer Graphics Forum*. Vol. 35. 7. Wiley Online Library. 2016, pp. 289–298.

[WK19] Rene Winchenbach and Andreas Kolb. “Multi-Level-Memory Structures for Adaptive SPH Simulations”. In: (2019).

[WRR18] Daniel Winkler, Massoud Rezavand, and Wolfgang Rauch. “Neighbour lists for smoothed particle hydrodynamics on GPUs”. In: *Computer Physics Communications* 225 (2018), pp. 140–148.

[Wu+18] Kui Wu et al. “Fast fluid simulations with sparse volumes on the GPU”. In: *Computer Graphics Forum*. Vol. 37. 2. Wiley Online Library. 2018, pp. 157–167.

[Xu16] Xiaoyang Xu. “An improved SPH approach for simulating 3D dam-break flows with breaking waves”. In: *Computer Methods in Applied Mechanics and Engineering* 311 (2016), pp. 723–742.

[Yan+09] He Yan et al. “Real-time fluid simulation with adaptive SPH”. In: *Computer Animation and Virtual Worlds* 20.2-3 (2009), pp. 417–426.

Appendix A

Scenario Visualization

In the figures in this appendix all finer particles, and hence of size $SMALL$, are blue, while the red particles are coarser and hence of size $LARGE$.

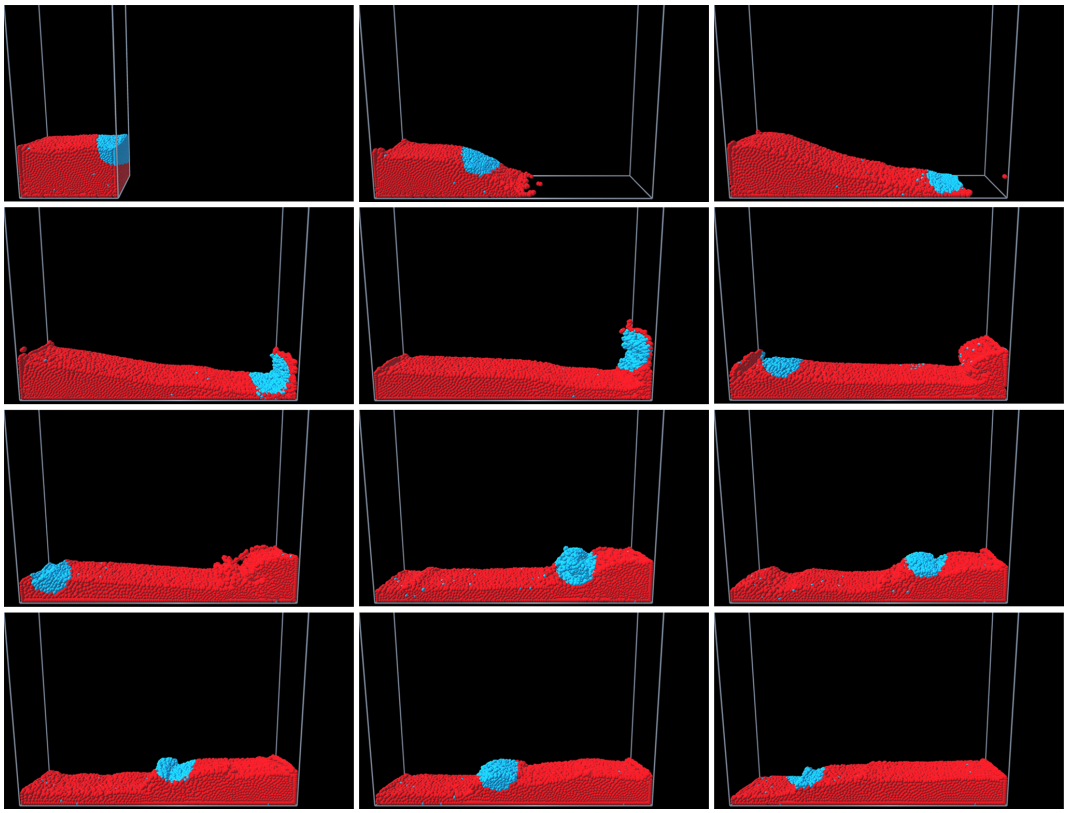


Figure A.1: Visualization of the dambreak scenario using the eye-tracking based adaptive method.

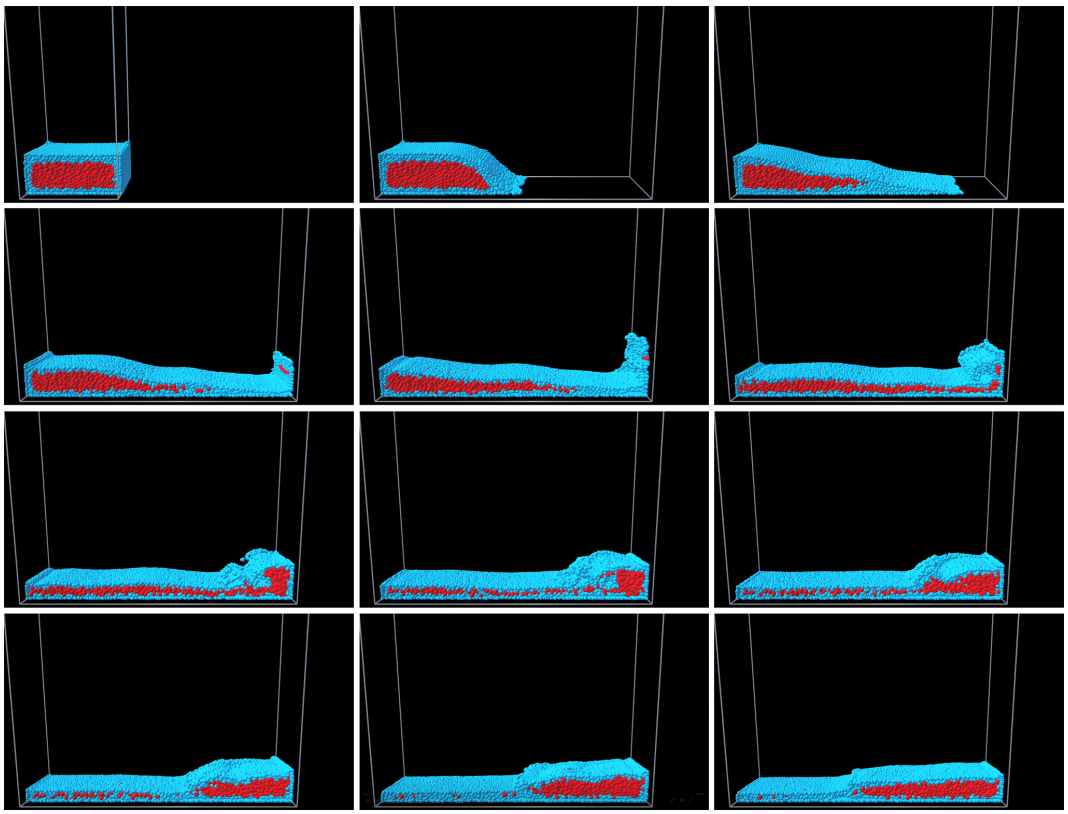


Figure A.2: Visualization of the dambreak scenario using the surface based adaptive method, where the visualization has been bisected to show how the particles are split and merged.

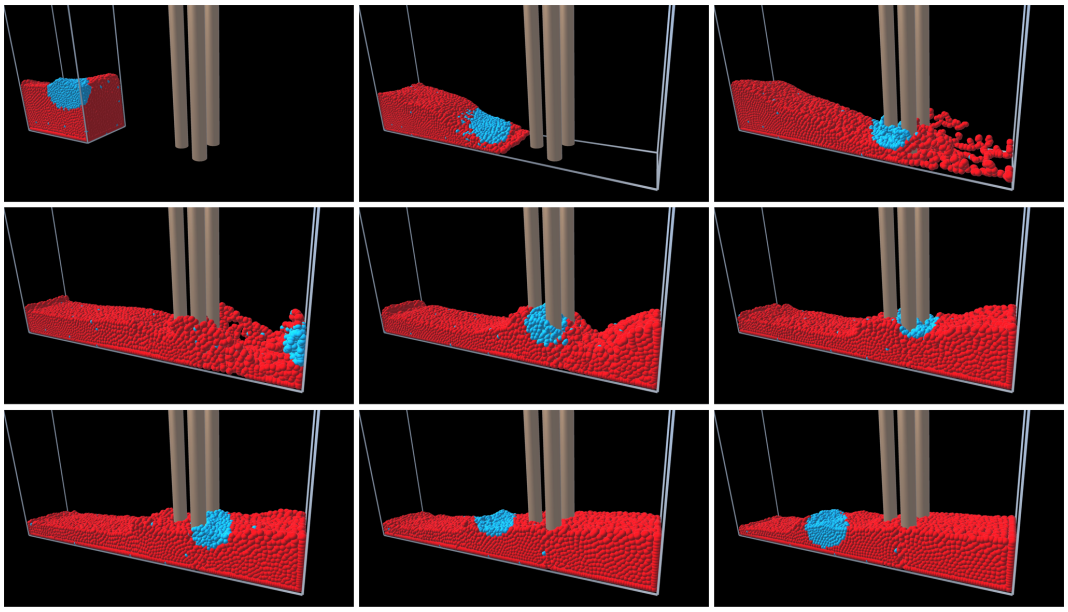


Figure A.3: Visualization of the dambreak scenario with collision using the eye-tracking based adaptive method.

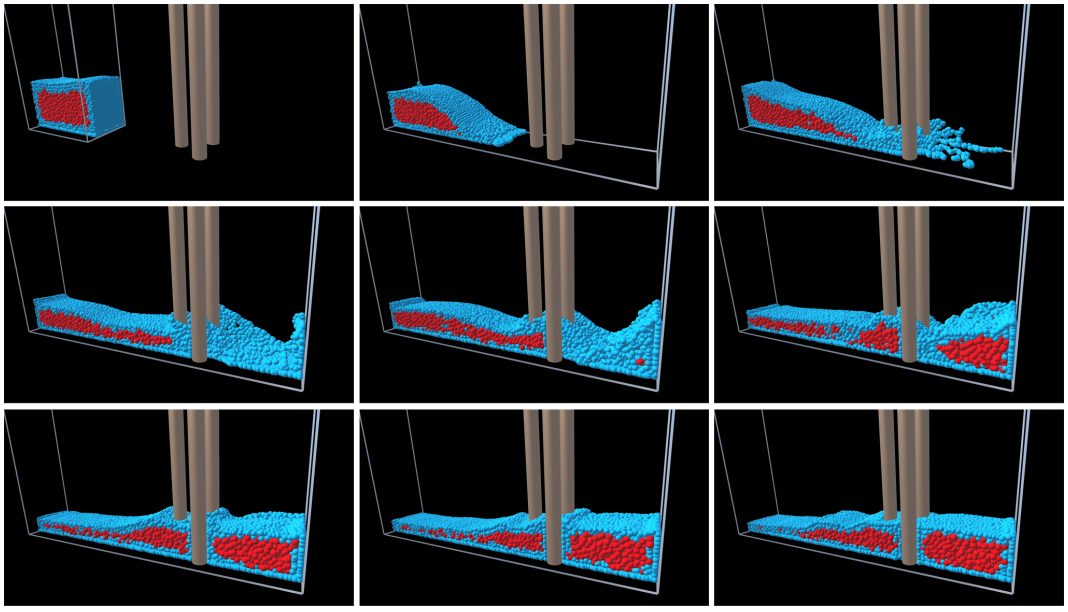


Figure A.4: Visualization of the dambreak scenario with collision using the surface based adaptive method, where the visualization has been bisected to show how the particles are split and merged.

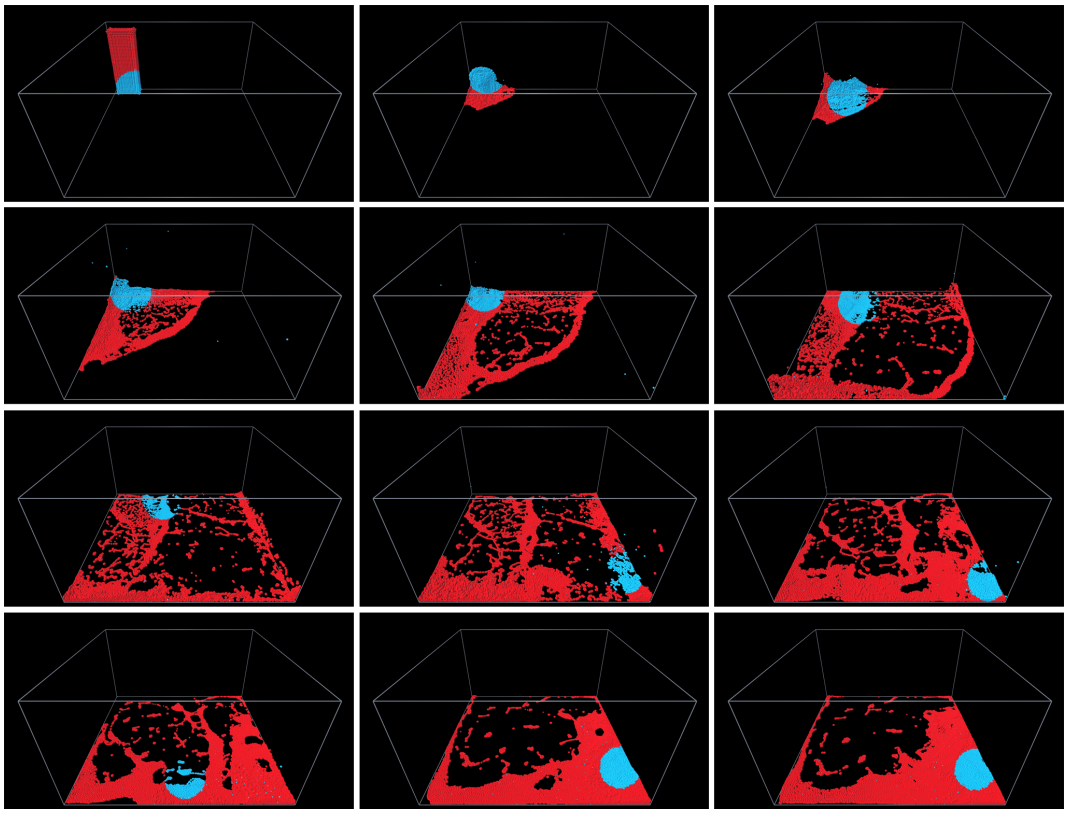


Figure A.5: Visualization of the scenario with a large bounding box to spread the particles out, using the eye-tracking based adaptive method.

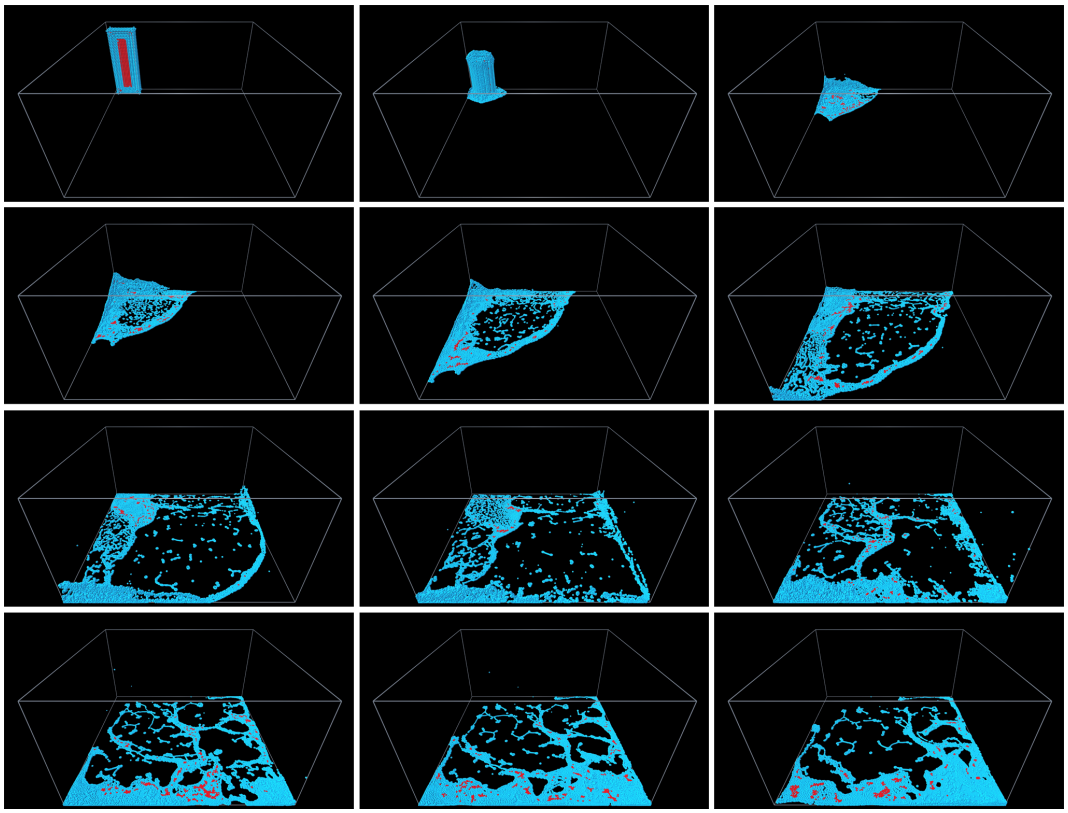


Figure A.6: Visualization of the scenario with a large bounding box to spread the particles out, using the surface based adaptive method.

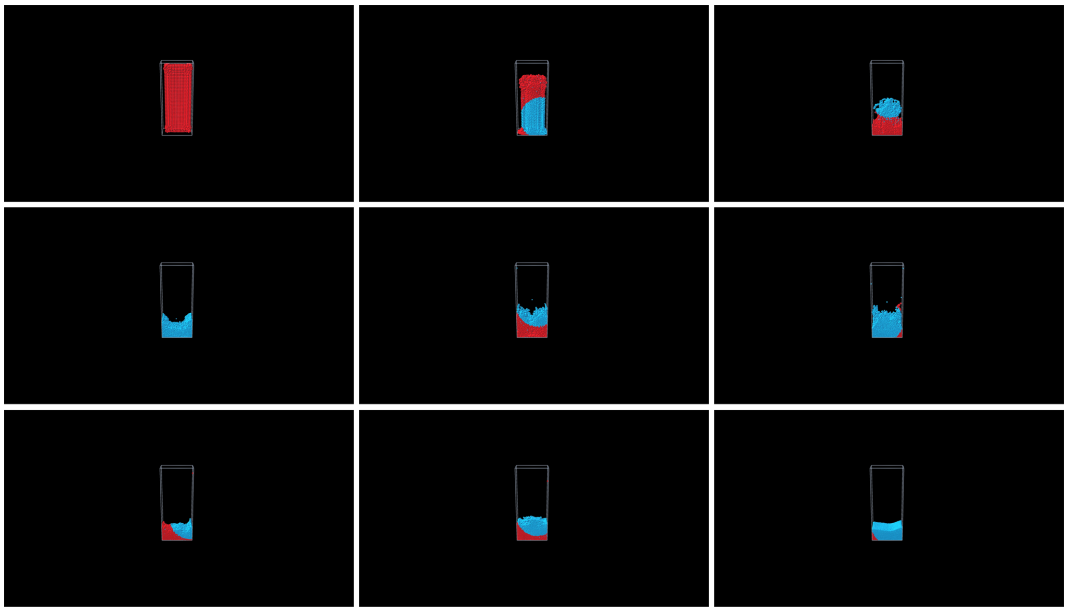


Figure A.7: Visualization of the scenario with a small bounding box to gather the particles, using the eye-tracking based adaptive method.

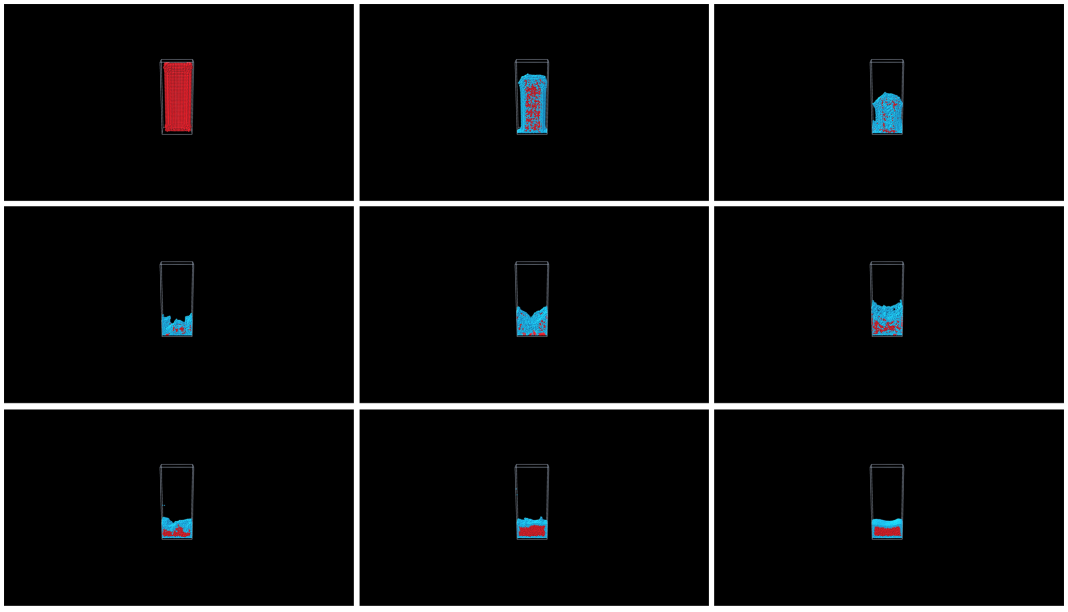


Figure A.8: Visualization of the scenario with a small bounding box to gather the particles, using the surface based adaptive method, where the visualization has been bisected to show how the particles are split and merged.

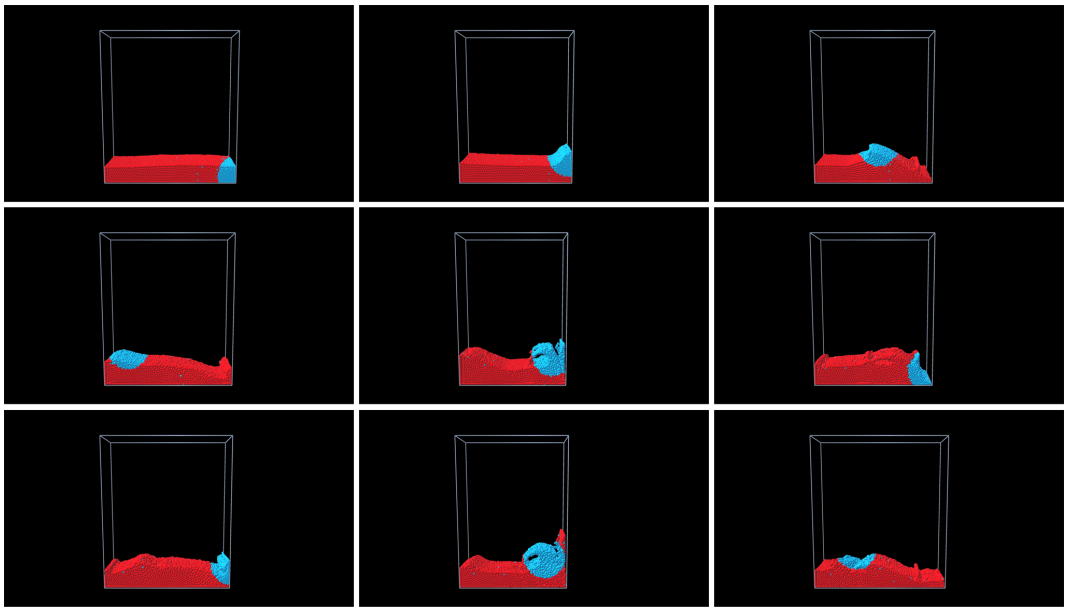


Figure A.9: Visualization of the scenario where waves are constantly generated, using the eye-tracking based adaptive method.

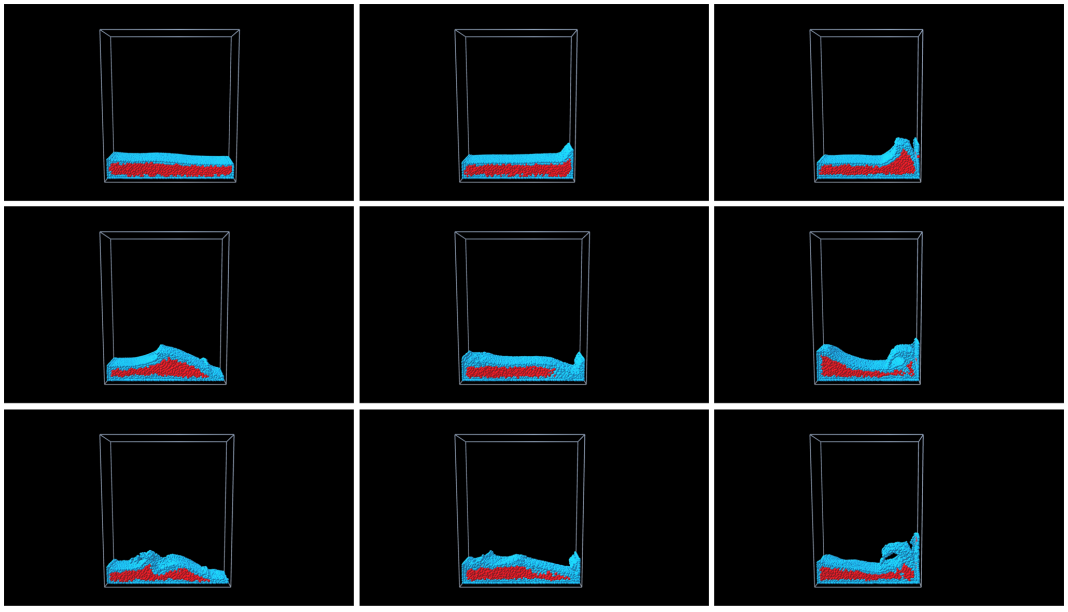


Figure A.10: Visualization of the scenario where waves are constantly generated, using the surface based adaptive method, where the visualization has been bisected to show how the particles are split and merged.