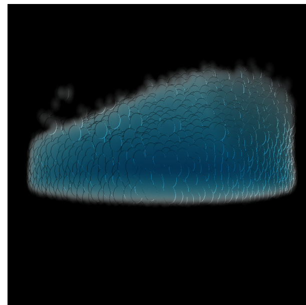
Linköping University

Fluid Simulation

Smoothed Particle Hydrodynamics on the GPU

Adam Alsegård (adaal265) Benjamin Wiberg (benwi631)
Emil Juopperi (emiju987) Jonathan Grangien (jongr713)
Simon Hedlund (simhe813)

November 24, 2016



Abstract

This report describes the physical concept of fluids as well as a mathematical model for fluids governed by the Navier-Stokes equations. The Smoothed Particle Hydrodynamics method (SPH) for simulating fluids is described, and implementation details of the method are explained. Numerical integration methods such as Euler and Leap-Frog integration are discussed.

The presented result is a program with support for real-time simulation and rendering of three-dimensional fluids. The properties of the fluid can be adjusted through a graphical interface, and the fluid particles can be rendered either as spheres or as an approximate fluid surface. The program is written in C++ with the SPH simulation implemented in OpenCL and rendering implemented in OpenGL.

Contents

1 Introduction **3**
1.1 Background . . . 3
1.2 Aim . . . 3

2 Fluid physics **4**
2.1 The Navier-Stokes equations . . . 4
2.2 Smoothed Particle Hydrodynamics . . . 5
2.2.1 Smoothing kernels . . . 6
2.2.2 Boundary conditions for smoothed particles . . . 6
2.2.3 Parameters . . . 6

3 Fluid simulation **8**
3.1 Numerical methods . . . 8
3.1.1 Euler integration . . . 8
3.1.2 Leapfrog integration . . . 8
3.2 Simulation . . . 8
3.2.1 Spatial partitioning . . . 8
3.2.2 Boundary conditions . . . 9
3.2.3 Multi-core simulation . . . 10
3.3 Rendering . . . 10
3.3.1 Direct particle rendering . . . 10
3.3.2 Point platting vs marching cubes . . . 11
3.3.3 Refraction and caustics . . . 11

4 Results **12**
4.1 Matlab . . . 12
4.2 C++ . . . 13
4.3 OpenCL . . . 13
4.4 Visuals . . . 13

5 Discussion **15**

6 Conclusion **16**

Bibliography **17**

Chapter 1

Introduction

1.1 Background

Simulating fluids, e.g. water, with computer graphics has been done with many different techniques in different contexts. Advanced techniques can produce very realistic looking fluids. This comes at a great cost of performance, and may not be possible in high-performance applications such as video games.

As technology and computing resources have advanced, some of the different approaches that have surfaced have become more common than others. One such approach is Smoothed Particle Hydrodynamics (SPH), and an implementation of this method is described in this report.

1.2 Aim

The aim of this project is to simulate water with high performance by using the SPH approach with calculations done on the Graphics Processing Unit (GPU). High performance in this context is to simulate a large number of particles with a high frame rate. A good enough frame rate in real-time for this project is 30 frames per second.

First an implementation of the method will be done in Matlab. When reasonable understanding of the method have been found the equations will be implemented in C++ with rendering on the GPU with OpenGL. The next step is to implement the equations in OpenCL in order to do the calculations on the GPU as well. The final step will be to render a surface of the water, have the particles integrate with a heightmap terrain and change parameters with a Graphic User Interface (GUI). If there is time an interaction with other objects will be implemented as well.

Chapter 2

Fluid physics

Simulation of a fluid's movement involves modelling several of its physical attributes such as viscosity, surface tension, pressure and incompressibility. A realistic enough behaviour means to be indistinguishable from that of an actual fluid.

To model physical attributes, one can make use of the Euler equations for fluid dynamics or the Navier-Stokes equations, the former being the simpler of the two.

2.1 The Navier-Stokes equations

The Navier-Stokes equations were put together in the 1800's by applying Newton's second law to fluid dynamics, and are fundamental in the modelling and analysis of fluids and fluid-like phenomena.

A well-known formulation is the Eulerian approach, which is a grid-based method. The equations describe the evolution of a set of fluid properties over time; the velocity field $\vec{v}$, the density field ρ and the pressure field p. There are many formulations of the Navier-Stokes equations depending on context and applied area, and the following equations are simplified for incompressible fluids.

The first equation (2.1) ensures mass conservation in the fluid:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{2.1}$$

The second equation (2.2), where $\mathbf{g}$ is an external force field (e.g. gravity), ensures conservation of momentum in the fluid:

$$\rho(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}) = -\nabla p + \rho \mathbf{g} + \nabla^2 \mathbf{v} \tag{2.2}$$

Since the increasing availability to solve equations numerically started taking form, different methods to implement these equations as a foundation for fluid simulation have surfaced. Particle systems were introduced as a technique in 1983 by Reeves [8], and since then both particle-based and grid-based approaches have been used.

2.2 Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) is a method that was initially developed for astrophysical simulations, but Müeller, Charypar & Gross [7] describe how the method can be applied to fluid simulation. SPH is a mesh-free Lagrangian method that simulates a fluid as a set of particles. The method is essentially an interpolation method, where a fluid property can be calculated anywhere in space by smoothing surrounding particles' properties.

The particles have a spatial distance, known as the smoothing length (R), which is a measure of how far a particle's properties can affect other particles. This means that when a property of a particle is calculated, only particles closer in proximity than the smoothing length contribute to it, as shown in Figure 2.1. Furthermore the smoothing kernels 2.2.1 weigh the contributions depending on the distance between the particles to be stronger for smaller distances.

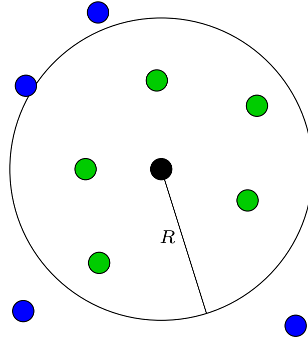


Figure 2.1: A property of the particle in the middle is calculated based on the green particles, the blue particles lie outside the radius of the smoothing kernel.

An advantage of the SPH approach is that the Navier-Stokes equations presented earlier can be greatly simplified. Since the number of particles is constant, and each particle has equal mass, the model inherently guarantees mass conservation, thus eliminiating (2.1). Additionally, the convection term $\mathbf{v}\cdot\nabla\mathbf{v}$ in (2.2) can be neglected according to Müeller et al.

The SPH method simplifies the Navier-Stokes equations into 3 forces acting on each particle; Pressure force, Viscosity force and External force. Since The Navier-Stokes equations are general for fluids it does not include consideration of surface tension forces. When simulating water this is an important factor for realistic behaviour and therefore a model for this is implemented beyond the Navier-Stokes equations.

The equation (2.3) is the general equation to evaluate any quantity A at position $\boldsymbol{r}$,

$$A_S(\boldsymbol{r}) = \sum_j (m_j \frac{A_j}{\rho_j} W(\boldsymbol{r} - \boldsymbol{r}_j, h)) \tag{2.3}$$

where m_j is the mass of particle j, A_j is the value of the quantity A for particle j and ρ_j is its density. W is a smoothing kernel (see chapter 2.2.1) that weighs the contribution from particle j

depending on its distance to the particle at position $\boldsymbol{r}$.

This general equation is the basis of all calculations of the particles' properties. Müeller et al. [7] describe how the equation can be modified to account for physical stability for the different fluid properties.

2.2.1 Smoothing kernels

Given a scalar distance between two particles and the smoothing length, a kernel determines the weight of the contribution from the particle. Depending on the quantity that is calculated, different properties are desired of the kernel. To accomplish correct calculations for the properties, the different specialized kernels proposed by Müeller et al. [7] are used.

2.2.2 Boundary conditions for smoothed particles

A physically based way of modelling boundaries is to have the boundaries contribute to the density of nearby particles as well as directly applying a boundary pressure force. A side-effect of the density contribution is that the effect of the boundary will propagate further from the boundary itself, since particles with high density apply higher pressure forces on nearby particles.

2.2.3 Parameters

The Navier-Stokes equations and the surface tension calculations include several parameters that manipulate the properties of the fluid. How each of them interacts and what kind of effect they have on the fluid is described here. Values given to the parameters need to be carefully considered and to study their effect on the fluid is necessary to give a simulated fluid of desired properties. The equations that contain these parameters are not included in this report and are general parameters for the SPH method [7].

- Gas constant
  This parameter affects the magnitude of the pressure forces in a way that a larger value results in stronger pressure forces. The fluid gets a more gas-like appearance when the magnitude of the pressure forces increases since the particles will not stay close together. A larger pressure force makes the particles keep a distance between each other and the volume of the particle system increases. When simulating water a low value, close to zero, is chosen for the gas constant. In conclusion this parameter will decide how gas-like the particle system behaves.

- Viscosity constant
  A fluid can be more or less viscous, which means how inclined particles are to be affected by the movement of particles close by. Increasing the viscosity constant makes the fluid look thicker and decreasing it gives the impression of a lighter fluid with less tendency of sticking together. Water is not thick and viscosity forces are not important for a realistic water simulation, hence a small value or zero is given to the viscosity constant.

- Surface tension constant
  A factor which the tension force is multiplied by and gives a linear increment in the magnitude of tension force.

- Threshold for surface tension
  Needed because when evaluating tension forces numerical problems can appear if certain properties in the equations are close to zero.

- Rest density
  Rest density works as an offset for the pressure field calculations. Mathematically it won't affect the pressure forces since they only depend on the gradients of the pressure field but with SPH smoothing kernels the offset will matter and makes the simulation numerically more stable.

Chapter 3

Fluid simulation

3.1 Numerical methods

3.1.1 Euler integration

Euler integration is a first-order integration method. The method involves updating the simulation states according to (3.1), where $\vec{x}_t$ is the state of the system at time instance t. The amount of change in the state is determined by its derivative, $\dot{\vec{x_t}}$, which is calculated based on the mathematical model of the system and its current state.

$$\vec{x}_{t+1} = \vec{x}_t + \dot{\vec{x}}_t \Delta t \tag{3.1}$$

The greatest advantage of the method is that only one state calculation has to be done for each timestep, i.e. to complete a true timestep only one state calculation has to be done. Its flaw is that the method has a first-order error, which means that the simulation needs a small timestep to ensure stability in the system.

3.1.2 Leapfrog integration

Leapfrog is an integration method that improves the simulation error to a second-order error, while still only needing one state calculation for each timestep[10]. The method, as the name implies, updates positions at whole timesteps $t = [0, 1, 2...]$ and velocities at non-integer timesteps $t = [1/2, 3/2, 5/2...]$.

3.2 Simulation

3.2.1 Spatial partitioning

The SPH method involves calculating the physical properties of each particle as a combination of the properties of the ones surrounding it. The smoothing kernels have a hard radius, outside of which the contribution to the property is zero. Therefore, each particle only depends on particles in its close proximity.

A naïve implementation of the property calculation would simply calculate a particle's property using the contribution from all particles in the simulation. Since all particles will have to access all other particles the algorithm has a complexity of $\mathcal{O}(n^2)$, where n is the number of particles.

Improvements to the naïve property calculation can be made by partitioning the particles spatially. There are many ways of spatial partitioning; binary space partitioning, k-d trees and octrees, but a method well-suited for SPH is a uniform grid.

Uniform grid

The simulation volume is divided uniformly into a three-dimensional grid. Each particle gets placed into a grid cell depending on its own position[2](slides 11-13). The cell size is as the radius of the smoothing kernel. This means that each particle only has to access all particles in its own and neighbouring grid cells to calculate a property, as shown in Figure 3.1. The complexity of the property calculation is now $\mathcal{O}(n * m)$ [7](ch. 5), where n is the number of particles and m is the number of grid cells.

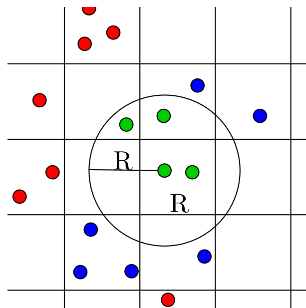


Figure 3.1: Radius and neighbouring grid cells for a given particle.

In 3.1 the blue and green particles are searched, resulting in identifying the green particles as contributing to its properties. The grid partitioning discards the red particles, greatly reducing the amount of particles needed to iterate through.

3.2.2 Boundary conditions

Constraint-based boundaries

Heightmap boundaries

The density- and force-based approach described in 2.2.2 has been implemented for simple geometry defined by a heightmap. A heightmap is 3D geometry defined by a 2D image, where each pixel's value represents the height of the geometry at that position. The two-dimensional nature of the heightmap implies that there are no holes in the geometry, i.e. no part of the geometry has geometry above or below it. A visualization of the heightmap is shown in Figure 3.2.

The implementation presumes that particles are affected by the geometrical boundaries only if they are within a set distance of its surface. The magnitude of the density and pressure force contributions are determined in a similar fashion to the smoothing kernels in 2.2, where the magnitude increases closer to the heightmap and approaches zero at a cutoff distance.

The density contribution from the heightmap on the particle is dependent on the distance d above the surface. The boundary pressure force F_p is applied in the direction of the heightmap's normal vector n at the point right below the particle.

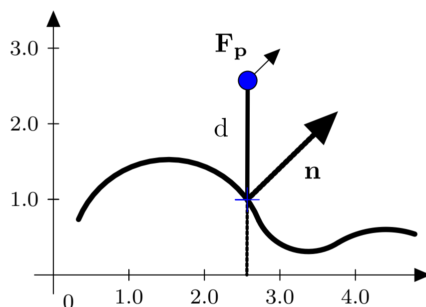


Figure 3.2: Heightmap

3.2.3 Multi-core simulation

The previously described SPH algorithm of simulating fluid can be divided into the following steps:

- Generate uniform partitioning grid
- Calculate particle densities
- Calculate particle forces
- Integrate particle states

The generation of the partitioning grid can be parallelized if special care is taken to ensure that no processing threads write to the same piece of memory at the same time. This is achieved through atomic write-locks to shared variables. All of the other steps can be executed in parallel internally, since the calculation for each particle only modifies the state of itself. This shows that the algorithm is well suited for a multi-core architecture. Harada, Koshizuka and Kawaguchi [5] describe ways of how to implement the simulation on GPUs.

3.3 Rendering

To be able to present the simulation a rendering of the particles has to be made. The particles can either be rendered individually or as a surface mesh, which is required for a more realistic look. There are many different methods of how to render particles as a water surface. This chapter describes and compares two of the most popular methods, marching cubes and point splatting.

The rendering was done on the GPU using OpenGL and GLSL coding languages.

3.3.1 Direct particle rendering

The easiest way to render particles in OpenGL are as *glPoints*. That way the particles will be rendered as squares with a user-defined number of pixels. A front-faced circle can be conceived from the square by discarding all pixels outside of a set radius.

The particles can also get a geometry by instancing, when the geometry is initialized by the user and then copied to all particles, or by generating one during run-time. If the geometry is generated during run-time either a tesselation shader or a geometry shader can be used. The tesselation shader is available with OpenGL 4.0 and above and is performance-wise the best way to generate a geometry [1]. The geometry shader has a memory limit that may limit the simulations and is

better used to change the geometry during run-time. This project has rendered spheres as point sprites and by using a tesselation shader.

3.3.2 Point platting vs marching cubes

To render the particles individually is good for visualization of their forces and movements but to get the whole picture a surface rendering is required. The traditional way to do this is by forming polygons of the particles at the surface and then divide them into a number of subsections. The normal of the surface is thereafter extracted and an iso surface is formed with the marching cubes method. However this is very computationally heavy and for a real-time application with ten thousand particles and above it is not preferable [6].

Instead a version of point splatting is used for this project where only the particles visible in the frame matters. The method is called *Screen space rendering with curvature flow* and is described theoretically by Simon Green [4]. He also describes the implementation of the method in Direct3D [3]. The first step of this method is to render the particles as spheres, either as point sprites or generated with a tesselation shader. Then a depth texture is generated from the camera's point of view. The next step is to blur this texture with the curvature flow algorithm to get a smooth surface of the depth. The surface's normal is then extracted from the depth texture. This is done by taking a step in x- and y-direction and forming vectors depending on the depth value in these pixels. The normal is the cross product of the x- and y-vector. The surface is then rendered with Phong shading with specular reflection based on Fresnel equations [4].

As a complement to the surface reflection, a thickness component is added. Additive blending is used to extract the thickness of the water. This component is then used for the color of the mesh; the higher the thickness, the bluer the water.

3.3.3 Refraction and caustics

Real water has different characteristics which are all more or less difficult to simulate. Examples of characteristics are shadows, refraction and caustics. Shadows can be added by using shadow mapping, which is to render a depth texture from the lights point of view and then use this texture's values to determine if the pixel is shadowed or not. Equations for refraction are possible to add but are computationally heavy to do in real-time [9]. There are different methods to implement both refraction and caustics in OpenGL. The simplest way to add caustics is to have a texture at the bottom of the water that reflects in the water. Implementation and simulation of all these characteristics are still subjects of research all over the world.

Chapter 4

Results

The theory of chapters 2 and 3 was implemented in the different steps described in 1.2. The results of these implementations are described in this chapter.

4.1 Matlab

A basic, partial implementation of the theory was implemented in Matlab. A low but sufficient amount of particles were simulated in two dimensions, with the SPH-based method proposed by Müeller et al. [7], including spatial partitioning with a grid. The performance was low but with first-hand assurance of the validity of the theory the Matlab implementation could be used as a reference for the C++ and the OpenCL implementation. 50 particles were simulated in Matlab with 6 frames per second, see Figure 4.1.

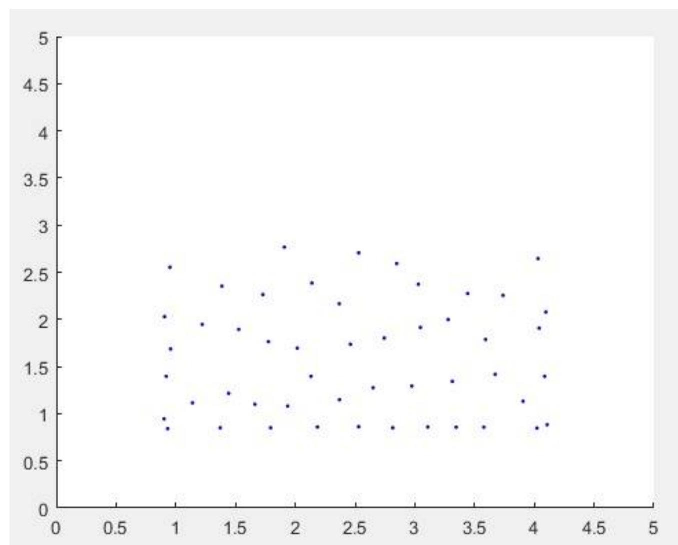


Figure 4.1: Particles simulated in Matlab

4.2 C++

A fluid simulation in three dimensions was made in C++, where the equations were implemented but not the grid system. A considerably improved performance over that of the Matlab implementation was observed, but not high enough for a polished result. Up to 500 particles were possible to simulate with 30 frames per second. Only individual rendering was used. No difference in performance was observed when the particles was rendered as point sprites or generated during run-time with the tesselation shader.

4.3 OpenCL

A high-performance fluid simulation was implemented in the multi-core computing framework OpenCL, shown in Figure 4.2. The multi-core implementation followed the reasoning in 3.2.3, and allowed for a great increase of particles over the C++ implementation. Up to approximately 20 000 particles were simulated at good enough frame rate when rendered as individual spheres.

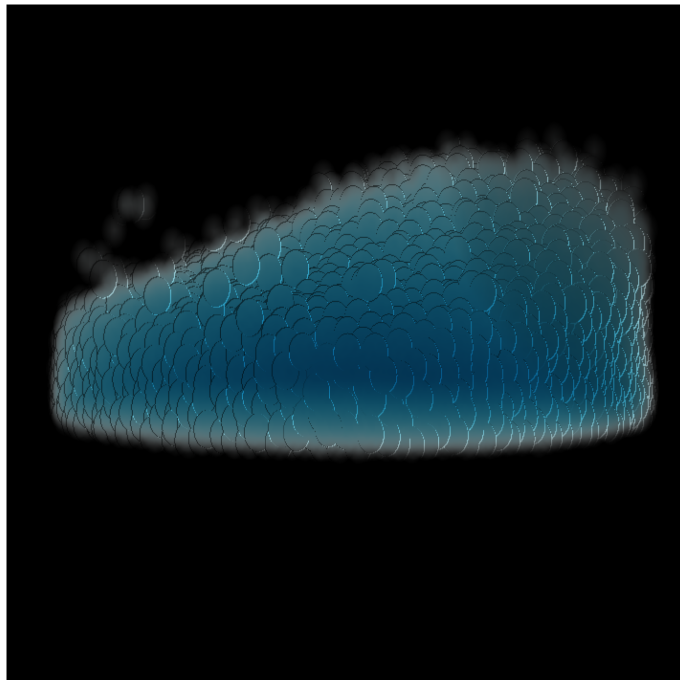


Figure 4.2: Particles simulated in OpenCL with depth and thickness.

4.4 Visuals

Rendering of the simulated fluid has been implemented in two different ways: as spheres and as a *free* surface. A *free* surface means that it takes into account free particles as splashes or drops and not just the big body of mass. The water surface rendering has been implemented as described in section 3.3.2 with reflection and thickness components. The implementation of the curvature flow algorithm still needs improvement for a solid surface to be rendered.

A heightmap with textures has been implemented and added to the scene. The boundaries of the heightmap does not affect the particles at this moment. A GUI where the user can change values of the parameters and the color of the background has been added as well, see Figure 4.3. The surface rendering was implemented separately from the heightmap and GUI. No performance loss was observed separately but after everything was assembled the frame rate dropped to below 20 frames per second and no more than 6000 particles could be used.

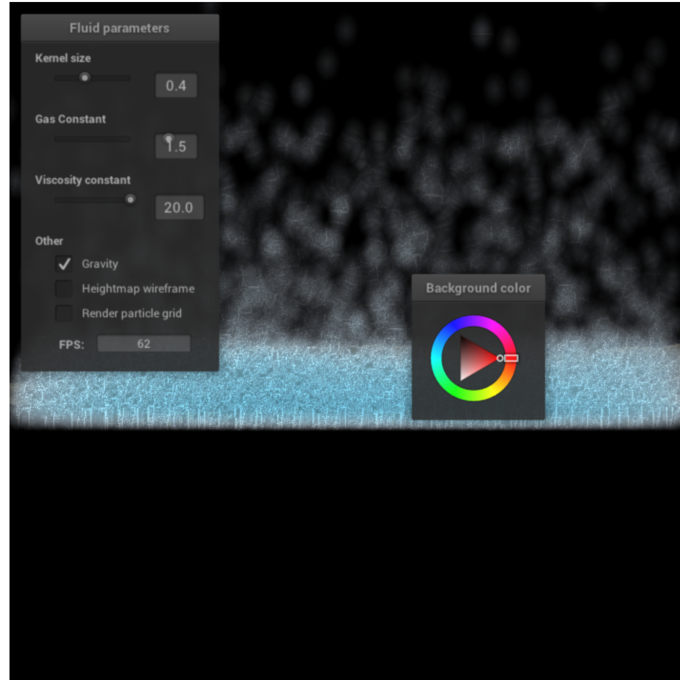


Figure 4.3: Simulation with blurred particles and GUI

Chapter 5

Discussion

The numerical integration method used in the implementations was Euler's method. It would have been better to implement the Leapfrog method (see section 3.1.2) and compare the results of the different methods to see which yielded a more visually pleasant result.

The implementation accounted for the stability problem by capping the delta time Δt to a maximum value. If the computer's performance was too poor, the simulation would run at a rate slower than real-time to ensure a small enough time step for numerical stability.

The values of the parameters used in the final results are largely unrealistic, despite working together in a way that yields realistic looking water. During a large part of the project's development the maximum amount of particles that was able to be rendered was very low, causing use of realistic parameters to result in unrealistic looking water. This is because a low amount of particles is not representative of reality. By the end result when higher amounts of particles could be rendered, realistic values were not tested as the visuals were deemed acceptable.

Some of the characteristics of the surface rendering have not been implemented yet, such as shadows, caustics and refraction. The rendering method used is for a free surface. A method for a non-free surface could have been faster but some realism would be lost.

The performance loss noted when the different visual part were assembled could be due to the addition of two computationally heavy algorithms, for the heightmap and the water surface respectively, and may be avoided by optimizing these algorithms. The decline of maximum particles possible is due to a memory limit set in OpenCL and could be optimized by changing this value, rewrite how the memory is allocated or remove components for the surface rendering completely.

Chapter 6

Conclusion

The SPH method as described in chapter 2 and 3 was well suited for the scope of this project. The implementations of the equations in C++ and OpenCL were successful and performance boosts were accomplished with every step. A visually pleasing result has been accomplished as well, which means that the aim of the project has been fulfilled.

Additional work could include implementation of other integration methods, more advanced components of the water surface, user interaction with the particles and spawning of new particles during run-time.

Bibliography

[1] M. Bailey (Oregon State Univeristy) *Tesselation Shader* (2016)

[2] E. Coumans (NVIDIA), *OpenCL Game Physics*, (2009)
http://www.nvidia.com/content/GTC/documents/1077_GTC09.pdf

[3] S. Green (NVIDIA), *Screen Space Fluid Rendering for Games*, From: Game Developers Conference (2010)

[4] S. Green, M. Sainz and W. J. van der Laan (NVIDIA), *Screen Space Fluid Rendering with Curvature Flow* In Proc. I3D 2009: The 2009 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pages 91-98 (2009)

[5] T. Harada, S. Koshizuka and Y. Kawaguchi, *Smoothed Particle Hydrodynamics on GPUs*
http://inf.ufrgs.br/cgi2007/cd_cgi/papers/harada.pdf (2007)

[6] K. Iwasaki, Y. Dobashi, F. Yoshimoto and T. Nishita, *Real-time Rendering of Point Based Water Surfaces* (2006)

[7] M. Müller, D. Charypar and M. Gross, *Particle-based fluid simulation for interactive applications*, In: Proc. of Sig- graph Symposium on Computer Animation (2003)

[8] W. T. Reeves, *Particle systems – a technique for modelling a class of fuzzy onjects*, ACM Transactions on on Graphics 2(2)

[9] T. Sousa *Generic Refraction Simulation*
http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter19.html. Fetched 14-03-2016.

[10] P. Young, *The leapfrog method and...*, University of Santa Cruz (2014)
http://young.physics.ucsc.edu/115/leapfrog.pdf. Fetched 14-03-2016.