Antonia Strantzi

MSc Computer Animation and Visual Effects

i7611774

Masters Project

Fluid Simulation Using Smoothed Particle Hydrodynamics (SPH)

National Centre for Computer Animation

Bournemouth University

22/08/2016

Contents

Abstract

Smoothed Particle Hydrodynamics (SPH) is a widely used method of simulating fluids in computer graphics. This paper explains this particular method, mentions previous research that has been done one it, as well as comparing it to other methods, especially the Fluid Implicit Particle (FLIP) method. Furthermore, an implementation of SPH is analysed. The implementation was done in C++ and the resulting simulation was imported to Houdini.

Chapter 1

Introduction

This paper follows the implementation of a Smoothed Particle Hydrodynamics solver in C++ and the presentation of work and research that has been done on this particular method.

Fluid simulations are a very sought after tool in today's graphics world. Widely used in games and films, they have become more and more in demand, therefore increasing the need for more accurate mathematical methods and representations.

There are two most common ways of representing fluids. Lagrangian and Eulerian methods, and sometimes hybrid ones that combine features from both methods. Lagrangian methods represent the fluid with particles, while Eulerian methods look at the fluid through a grid structure that stores information about the fluid's motion. Hybrid methods are also used, because they combine the best parts of the previously mentioned representations. The most popular Langrangian method is SPH, which is the one implemented for this paper, while a very used Eulerian method is the Staggered Marker and Cell Grid (MAC Grid). Lastly, there are hybrid methods, such as the Fluid Implicit Particle (FLIP) and the Particle in Cell (PIC). FLIP is the most popular method used in today's graphics.

SPH is a particle-based method used to simulate fluids. The particles making up the volume of the fluid have attributes like velocity, acceleration and mass, which are the common attributes in all particle systems[10]. In SPH, additional attributes are needed, such as density, pressure and forces affecting the motion[10].

The implementation of the SPH solver was done in C++ and the simulation was visualized using the OpenGL[3] and ngl[4] libraries. Furthermore, the simulation was also exported to Alembic[5] format and imported to Houdini[6] so that a scene could

be rendered out.

At the end, the results of the simulation are analysed and problems that rose during the process are also discussed.

Chapter 2

Fluid Simulation

2.1 Previous Work

Over the last few decades, fluid behaviour has been a fascinating topic for researchers. There have been hundreds of publications, papers and lectures on the subject, as well as a lot of different approaches.

One of the first fluid simulation methods was developed back in the 1960's by Harlow. He led a group in the Los Alamos National Laboratory and developed the particle-in-cell. The initial aim of the group was to develop numerical techniques that would aid the USA's national defense system[26]. The PIC method is still used to this day, albeit more in hybrid methods.

In 1977, Gingold and Monaghan, and Lucy, invented smoothed particle hydrodynamics in order to simulate nonaxisymmetric phenomena in astrophysics[27]. They found out that SPH suited their simulation needs and could be extended to other physical phenomena. Since, then several approaches have been by researchers and scientists, most of which focused on perfecting particular aspects of the technique. In 1996, Desbrun and Gascuel proposed an alternative method of calculating the pressure force, by combining the ideal gas law with the rest density, and therefore symmetrizing the pressure force. In 2003, Muller et al.[16] proposed a model for calculating the fluid's surface tension by using colour fields. Spatial hashing was a big step to the optimization of the method, since it provided a quick solution to the neighbour search problem. It was developed by Teschner et al. in ETH, Zurich[20]. Micky Kelager's project from 2006[13], is a very popular citation when it comes to SPH, since he includes a lot of the new methods and offers a clear insight into the SPH process. This is the paper that was mostly followed during this implementation.

2.2 Fluid Simulation Theory

According to Englesson (2011), in Lagrangian representations, the fluid is approached as a collection of particles, or atoms, which make up the fluid's volume. Langrangian methods have advantages in that they are understandable and all calculations are performed on the particles. On the other hand, though, the calculations rely heavily on the particles' density, therefore making the calculations, in areas with low density, inaccurate[1].

This problem is not present when using Eulerian methods. For this kind of methods, the fluid is observed through a number of cells that make up a grid. Because of this, Eulerian representations are also called grid-based representations. The calculations are performed on the cells of the grid, with no subsequent mathematical “gaps” in low density areas. The disadvantage of Eulerian simulations is the fact that they need a very big number of cells in order for the simulation to be as realistic as possible. This number can easily rise to the millions, thus reaching high computational complexity[1].

Hybrid methods provide a middle ground where we can still have the advantages of particles, but inside a grid, which provides higher accuracy[1].

Three methods from each category are explained in the following pages.

2.3 Navier-Stokes Equations

The two fundamental equations that describe a fluid’s motion are called Navier-Stokes equations, and they are used to describe the fluid’s velocity field over time. The two equations are[10]:

$$\nabla \cdot \mathbf{u} = 0 \quad (1)$$

$$\frac{\partial \mathbf{u}}{\partial x} = -\mathbf{u} \cdot \nabla \mathbf{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{F} \quad (2)$$

The first equation ensures that the velocity field has zero divergence, therefore conserving mass. This comes from the Helmholtz-Hodge Decomposition, which

states that any vector field $\mathbf{w}$ can be decomposed into the following form:

$$\mathbf{w} = \mathbf{u} + \nabla q$$

Where $\mathbf{u}$ is a vector field whose divergence needs to be zero and q is a scalar field. This shows how any vector field is the sum of a mass conserving field and a gradient one[11]. In other words, equation (1) says that the amount of fluid flowing into our predefined volume must be equal to the amount of fluid that's flowing out.
Equation (2) calculates the velocity field over time and sums up the following terms[12]:

$-\mathbf{u} \cdot \nabla \mathbf{u}$: the self-advection term, showing how the divergence affects the velocity

$-\frac{1}{\rho}\nabla p$: the pressure term, which models how the particles move away from high pressure areas

$v\nabla^2 \mathbf{u}$: the viscosity term, which captures the relationship between the velocity and the viscosity variable v. The highest this variable is, the more viscous, namely thicker, the fluid is.

$\mathbf{F}$: any external forces that are applied to the fluid, like gravity

2.4 Popular Methods

2.4.1 Marker and Cell Grid

The Marker and Cell (MAC) Grid method was introduced by Harlow and Welch in 1965[17]. They introduced a new way of tracking fluid motion through particles, called marker particles and grid, called staggered grid. A staggered grid is different from regular ones in that it stores quantities and their components in different parts of its cells[1]. Particularly, in a 3D grid, the three components of each cell's velocity vector are stored on the cell faces, while the pressure is stored in the cell's centre. Storing the velocity on the cell faces provides more stable simulations, instead of storing it at the centre. Besides the grid cells, marker particles are used to represent the fluid volume and determine which cells contain fluid and which ones don't[10]. The following image illustrates a 3D MAC grid cell:

2.4.2 Fluid Implicit Particle

The Particle-in-Cell method was first described in Harlow's 1962 paper[23]. The main idea is that the quantities are stored in the particles and then transferred onto the grid and then, after these quantities are updated, back onto the particles. Bridson[24] explains that all the quantities, velocity included, are transferred to the grid. Then, all the non-advection terms, such as gravity acceleration, pressure, etc. are calculated on the grid, as with purely Eulerian representations. Then, using trilinear interpolation (or bilinear, if the simulation is in 2D), the quantities are transferred back to the particles, which are then advected in the grid velocity field. The FLIP method is a variation of PIC and was developed by Brackbill and Ruppel in 1986[31]. The difference from PIC is that the particle quantities, after being calculated on the grid, are not transferred back to the particles as they are, but they are added to the particles' previous values.

3. Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics is a method first introduced in 1977 by Gingold and Monaghan, and Lucy[8][9], to simulate astrophysical phenomena. According to a review on the theory and application of SPH by Monaghan (2005)[7] "smoothed particle hydrodynamics is a method for obtaining approximate numerical solutions of the equations of fluid dynamics by replacing the fluid with a set of particles. For the mathematician, the particles are just interpolation points from which properties of the fluid can be calculated. For the physicist, the SPH particles are material particles which can be treated like any other particle system."
According to Kelager[13] "SPH is an interpolation method to approximate values and derivatives of continuous field quantities by using discrete sample points". These sample points are represented by particles that carry certain properties.

3.1 Particle Systems

Particle systems have been widely used nowadays, in order to create convincing simulations of fuzzy phenomena, either related to real world situations or make-believe ones, usually seen in motion pictures and games. The first major publication on particle systems came with William T. Reeves' paper (1983)[28] where he defines particle systems as: "a method for modeling fuzzy objects such as fire, clouds, and water. Particle systems model an object as a cloud of primitive particles that define its volume. Over a period of time, particles are generated into the system, move and change form within the system, and die from the system." Based on this paper, Reeves created a wall of fire effect in Star Trek: Wrath of Khan (1982)[29]. Fuzzy objects do not consist of well-defined and rigid surfaces, but irregular and complex ones[28]. They do not take up a specific amount of space and they are not constrained within strict boundaries.

3.1.1 Particle System Properties

In every particle system, each particle has a specific set of attributes. These attributes are[30]:

1. Initial position
2. Initial velocity
3. Initial size
4. Initial colour
5. Initial transparency (alpha)
6. Shape
7. Lifespan

The subsequent movement of the particle is defined by calculating its velocity, using the acceleration parameter. The acceleration is usually defined by the gravity field using Newton's Second Law $\mathbf{F} = m\mathbf{a}$ where *m* is the particle's mass.

3.2 SPH Particle Properties

Besides the usual attributes that particles carry within a particle system, SPH particles carry further physical quantities, like mass-density, pressure and temperature[13].

3.3 Definition

As mentioned above, SPH is an interpolation method that uses kernels. These kernels model a delta function, according to the particles' positions. This integral interpolant of any quantity function $\mathrm{A}(\mathbf{x})$ is given by the following formula[13]:

$$A_I(\mathbf{r}) = \int_{\Omega} A(\mathbf{r}')W(\mathbf{r} - \mathbf{r}', h)d\mathbf{r}'$$

Where Ω is the space in which the interpolation is calculated, W is a smooth kernel weighting function with h as its width. The numerical equivalent to the integral is a summation interpolant[15]:

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

In fluid particle terms, $A(\mathbf{r})$ represents an attribute of a certain particle in position $\mathbf{r}$, m is the mass and ρ is the particle's mass-density.
When applying a gradient and a Laplacian operator to $A(\mathbf{r})$, the following equations occur:

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

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

Therefore, the gradient and Laplacian operators are only applied to the kernel function, because, according to the rules of differentiation, the $m_j \frac{A_j}{\rho_j}$ part equals to zero when differentiated.

3.4 Smoothing Kernels

A smoothing kernel is a weighting function which adjusts the particles' quantities according to the distance among them[14]. There are different smoothing kernels that one can use for SPH approximations, depending on the quantity that needs to be calculated. The two properties a smoothing kernel must have are[13]:

$$\int_\Omega W(\mathbf{r}, h) d\mathbf{r} = 1$$

$$\lim_{h \to 0} W(\mathbf{r}, h) = \delta(\mathbf{r})$$

Furthermore, the kernel function must always be positive, and even. Namely:

$$W(\mathbf{r}, h) \geq 0$$

$$W(\mathbf{r}, h) = W(-\mathbf{r}, h)$$

In this implementation three types of kernels are used, the 6th polynomial, the spiky and the viscosity kernel, which will be explained in later chapters.

4. Implementation

For the implementation, a certain amount of particles was used, that stays fixed throughout the duration of the simulation, therefore conserving the fluid's mass.

4.1 Navier-Stokes for SPH

Given the fact that the amount of particles stays the same for the duration of the simulation, and the mass also stays fixed, the first Navier-Stokes equation is redundant, since mass conservation is already guaranteed. Furthermore, since we are talking about Lagrangian fluid dynamics, and not Eulerian representations, the advection term of the Navier-Stokes equations is redundant as well. This is due to the fact that in Langrangian representations, it's the particles that define the fluid, and any field quantity only depends on time[13]. In conclusion, for Langrangian representations, the Navier-Stokes equations become:

$$\frac{\partial \mathbf{u}}{\partial x} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{F}$$

4.2 Mass and Mass-Density

4.2.1 Mass

All the particles have an equal mass, that stays fixed throughout the simulation. The mass is one of the most important attributes, since it is used to calculate the density and the forces applied. Priscott[14], based on Kelager[13] uses the following equation to calculate the mass of the particle, using the particle's density, the fluid's volume and the number of particles making up the volume:

$$m = \frac{V}{\rho}n$$

However, the mass of the particles in this implementation is set to *0.02*, because the above equation resulted in large particle masses that brought instability to the

calculations. The volume of the fluid is set as a global parameter and its shape starts off as either a sphere or a box.

4.2.2 Mass-Density

Mass-density is a vital term to have in order to make calculations on the particles. In contrast to the mass, which can be user-defined, mass-density is a continuously changing field, which needs to be calculated on every iteration of the simulation.

For the mass-density computation we have to use a smoothing kernel function. For this computation Kelager[13] uses the 6th polynomial kernel as the default one.

The kernel is given by:

$$W_{poly6}(\mathbf{r}, h) = \frac{315}{64\pi h^9}\begin{cases}(h^2 - \|\mathbf{r}\|^2)^3, & 0 \le \|\mathbf{r}\| \le h \\ 0, & \|\mathbf{r}\| > h\end{cases}$$

The gradient of the kernel is:

$$\nabla W_{poly6}(\mathbf{r}, h) = -\frac{945}{32\pi h^9}\mathbf{r}(h^2 - \|\mathbf{r}\|^2)^2$$

The Laplacian of the kernel is:

$$\nabla^2 W_{poly6}(\mathbf{r}, h) = -\frac{945}{32\pi h^9}(h^2 - \|\mathbf{r}\|^2)(3h^2 - 7\|\mathbf{r}\|^2)$$

The mass-density is computed using the SPH approximation:

$$\rho_i = rest\_density + \rho(\mathbf{r}_i)$$

$$\rho_i(\mathbf{r}) = rest\_density + \sum_j m_j \frac{\rho_j}{\rho_j} W_{poly6}(\mathbf{r}_i - \mathbf{r}_j, h)$$

Which turns into:

$$\rho_i(\mathbf{r}) = rest\_density + \sum_j m_j W_{poly6}(\mathbf{r}_i - \mathbf{r}_j, h)$$

where j represents each particle in the "neighbourhood" of the particle whose mass-density is currently being calculated. The method of finding the particle's neighbourhood is called spatial hashing and it is going to be explained in detail in a later chapter.

4.3 Internal Forces

4.3.1 Pressure

The pressure is computed by using the ideal gas law[13]:

$$pV = nRT$$

$V = \frac{1}{\rho}$ is the volume per unit mass, n is the amount of particles inside the volume, R is the universal gas constant and T is the temperature. If the temperature remains stable, then the right hand side of the equation is a constant and can be replaced with a gas stiffness constant k depending only on the number of particles used. After replacing the right hand side with k, then the equation becomes:

$$pV = k \;\Rightarrow\; p\frac{1}{\rho} = k \;\Rightarrow\; p = k\rho$$

According to Desbrun and Gascuel (1996)[18] liquids should have a constant mass-density at rest. Therefore, the liquid should have some internal cohesion, resulting in attraction-repulsion forces. For that reason, we use a modified version of the ideal gas law:

$$(P + P_0)V = k$$

where Po is an additional rest pressure: $P_0 = k\rho_0$, where ρo is the rest density of the fluid material.

4.3.2 Pressure Force

After calculating the pressure for each particle, the SPH approximation is used for the pressure force:

$$\mathbf{f}^{pressure} = -\nabla p(\mathbf{r})$$

$$\mathbf{f}_i^{pressure} = -\sum_{i \neq j} p_j \frac{m_j}{\rho_j} \nabla W_{spiky}(\mathbf{r}_i - \mathbf{r}_j, h)$$

This equation, though, does not provide a symmetrical pressure force, since the particle only uses the other particles' pressures to compute its pressure force. Consequently, this does not conserve the action-reaction law[13]. Symmetrizing the pressure force, yields the following formula[18]:

$$\mathbf{f}_i^{pressure} = -m \sum_{i \neq j} m_j \left(\frac{p_i}{{\rho_i}^2} + \frac{p_j}{{\rho_j}^2} \right) \nabla W_{spiky}(\mathbf{r}_i - \mathbf{r}_j, h)$$

Another faster and stable solution to symmetrizing the pressure force, proposed by Muller et al. (2003), is:

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

A smoothing kernel function is also needed to calculate the pressure force. The default 6th polynomial is not used in this case, but a spiky kernel is preferred. The spiky kernel is more suitable because when it comes to pressure force, particle clusters need to be created in a more explicit and precise way[13].

The spiky kernel yields:

$$W_{spiky}(\mathbf{r}, h) = \frac{15}{\pi h^6} \begin{cases} (h - \|\mathbf{r}\|)^3, & 0 \leq \|\mathbf{r}\| \leq h \\ 0, & \|\mathbf{r}\| > h \end{cases}$$

The gradient of the kernel is:

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

The Laplacian of the kernel is:

$$\nabla^2 W_{spiky}(\mathbf{r}, h) = -\frac{90}{\pi h^6}(h - \|\mathbf{r}\|)(h - 2\|\mathbf{r}\|)$$

4.3.3 Viscosity

Viscosity is the tendency of a fluid to resist flow. The more viscous the fluid, the more the molecules stay together. For instance, water is a low viscosity fluid, while honey has higher viscosity. The coefficient μ defines the viscosity strength[13]. The SPH approximation for the viscosity term is:

$$\mathbf{f}_i^{viscosity} = \mu \nabla^2 \mathbf{u}(\mathbf{r}_i)$$

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

This term, like the pressure force term, is asymmetric. Muller[16] has proposed a symmetrized equation, not using absolute velocities, but velocity differences, which are the ones that viscosity forces depend on.

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

Since we don't want the forces due to viscosity to increase the relative velocity and destabilize the system, the Laplacian operator of the smoothing kernel needs to be positive. A kernel that achieves this is the following one[13]:

$$W_{viscosity}(\mathbf{r}, h) = \frac{15}{2\pi h^3} \begin{cases} -\frac{\|\mathbf{r}\|^3}{2h^3} + \frac{\|\mathbf{r}\|^2}{h^2} + \frac{h}{2\|\mathbf{r}\|} - 1, & 0 \leq \|\mathbf{r}\| \leq h \\ 0, & \|\mathbf{r}\| > h \end{cases}$$

The gradient of the kernel is:

$$\nabla W_{viscosity}(\mathbf{r}, h) = \frac{15}{2\pi h^3} \mathbf{r} \left(-\frac{3\|\mathbf{r}\|}{2h^3} + \frac{2}{h^2} - \frac{h}{2\|\mathbf{r}\|^3} \right)$$

The Laplacian of the kernel is:

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

4.4 External Forces

The external forces are applied after all the internal forces are calculated. All the external forces are combined into a sum.

4.4.1 Gravity

The gravity force is one that always has to be taken into account in particle systems. Given g gravitational acceleration, where g = (0,-9.8,0), the gravitational force is[13]:

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

4.4.2 Surface Tension

Inside the fluid volume, the particles are subject to attractive forces from neighbouring particles. Inside the fluid these forces are equal in every direction, but for particles close to the surface, these forces are unbalanced. Attraction forces among the particles act in the direction of the surface normal of the particle, and they also depend on the curvature of the surface, with higher curvature spots, applying higher force[16]. Muller et al. (2003) explain how the surface tension is calculated.

The fluid "area" can be tracked using a colour field, which is 1 inside the fluid, and 0 outside it[19]. The smoothed colour field is:

$$c_S = c_S(\mathbf{r})$$

$$c_S = \sum m_j \frac{1}{\rho_j} W_{poly6}(\mathbf{r}_i - \mathbf{r}_j, h)$$

The surface normal field is found by applying a gradient operator on the colour field:

$$\mathbf{n} = \nabla c_S$$

$$\mathbf{n} = \sum \frac{m_j}{\rho_j} \nabla W_{poly6}(\mathbf{r}_i - \mathbf{r}_j, h)$$

The curvature of the surface depends on the divergence of the surface normal field:

$$\kappa = \frac{-\nabla^2 c_S}{\|\mathbf{n}\|}$$

The surface tension is calculated using the surface traction $\mathbf{t}$ and a normalized scalar field $\delta = \|\mathbf{n}\|$ which is positive only when the surface normal length exceeds a certain value l, otherwise numerical errors may occur. The surface traction is:

$$\mathbf{t} = \sigma\kappa \frac{\mathbf{n}}{\|\mathbf{n}\|}$$

Where σ is the tension coefficient that depends on the fluids making up the surface[13].

The surface tension is computer using the following equation, only when $\|\mathbf{n}\| \geq l$:

$$\mathbf{f}^{surface} = \sigma\kappa\mathbf{n} = -\sigma\nabla^2 c_S \frac{\mathbf{n}}{\|\mathbf{n}\|}$$

4.4.3 Buoyancy

Buoyancy needs to be taken into account when we want to simulate gaseous fluids. Kelager explains that buoyancy is caused by diffusion of temperatures, but when modeling isothermal fluids, an artificial buoyancy can be given using the following equation[13]:

$$\mathbf{f}^{buoyancy} = b(\rho - \rho_0)\mathbf{g}$$

Where b is a positive parameter representing the artificial buoyancy diffusion. In non-gaseous fluids, like water, parameter b is set to zero.

4.4.4 User Interaction

If a user can manipulate the motion of the fluid, then an extra external force is added to the sum of forces. The user may have the ability to control the fluid itself, its boundaries or throw obstacles in it. An example would be for the user to be able to block the fluid's movement by clicking on the fluid on-screen.

4.5 Acceleration

The acceleration derives from Newton's second law of motion: $\mathbf{F} = m\mathbf{a}$, where the sum of all the forces is divided by the density.

$$\mathbf{a} = \frac{\mathbf{F}}{\rho}$$

4.6 Velocity

For the update of the velocity and then the position of the particle, the implicit Euler scheme is used, as described by Kelager (2006)[13]:

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

4.6.1 XSPH Velocity

Priscott (2010) implements a particle velocity correction technique called XSPH, in order to correct velocities that divert too much from an average amount. The equation used is the following[14]:

$$\mathbf{u}_i = \mathbf{u}_i + \varepsilon \sum_j \frac{2m_j}{\rho_i + \rho_j} W_{poly6}(\mathbf{r}_i - \mathbf{r}_j, h)$$

In this implementation, the ε parameter is set to 0.1 so that the velocities don't change too much.

4.7 Spatial Hashing

A vital part of a Langrangian fluid simulation process is the neighbour search. All SPH approximations are performed while taking into account the attributes of particles that are located close to the particle in question.

When using thousands of particles, searching for neighbours for each particle among all the others, will be extremely time-costly, since the computational complexity is $O(n^2)$, where n is the number of particles. That is why optimization methods are needed. In this case, a method called "spatial hashing" is used. Spatial hashing is a fast nearest neighbour search algorithm and has a computational complexity of $O(1)$[13].

The way spatial hashing works is that every particle's position is hashed to create a key. This key specifies the cell in which the particle's position is going to be stored in the hash table. The hash function being used in this algorithm, assigns the same key to particles located close to each other. Therefore, in order to find a particle's neighbours, all one has got to do is look up the hash table and get a list of the particles hashed in the same cell.

Teschner (2003)[20] has proposed the following hash function:

$$hash(\mathbf{r}) = \left(\hat{\mathrm{x}}(\mathbf{r}_x)p_1 \; xor \; \hat{\mathrm{x}}(\mathbf{r}_y)p_2 \; xor \; \hat{\mathrm{x}}(\mathbf{r}_z)p_3\right) \bmod n_H$$

The numbers $p1$, $p2$ and $p3$ are large prime numbers[13]:

$$p_1 = 73856093$$
$$p_2 = 19349663$$
$$p_3 = 83492791$$

Kelager (2006) describes the discretizing $\hat{\mathrm{x}}$ function, which takes a vector of floating values and converts it to a vector of integers by rounding down the numbers, according to a cell size l. Here, this cell size is assigned the smoothing length factor h[13].

$$\hat{\mathrm{x}}(\mathbf{r}) = \left(\left\lfloor\frac{\mathbf{r}_x}{l}\right\rfloor, \left\lfloor\frac{\mathbf{r}_y}{l}\right\rfloor, \left\lfloor\frac{\mathbf{r}_z}{l}\right\rfloor\right)$$

Lastly, n_H is the hash table size, a large prime number that is large enough to avoid collisions among the positions hashed. Therefore, n_H is defined as:

$$n_H = next\_prime(2n)$$

The prime function returns the next prime number after $2n$, where n is the number of particles.

Every time the particles' positions are updated, the hash table needs to be updates as well, in order for the new positions to be hashed. The equation used to fill the hash table for a particle *p* is[13]:

$$hashmap\big[hash\big(\hat{\mathrm{x}}(\mathbf{r})\big)\big] = p$$

When a particle *p*'s neighbours need to be found, all there is to be done is retrieve a list of the cells hashed to the same key as *p*. However, it is possible that not all particles located in close proximity are going to be hashed in the same cell.

For this reason, there needs to be an extra search for particles located around particle *p*. Priscott (2010) describes a process of defining a bounding box around the particle[14]. Then, we iterate over discretized positions (using the $\hat{\mathrm{x}}$ function) inside this box and retrieve their hash keys. For each hash key retrieved, we get the positions hashed to that key and add them to a dynamic list. However, the particles added to that list are not always guaranteed neighbours. Therefore, a final check needs to be performed, so that the particles *j* in that list lie within the smoothing length radius *h* around particle *p*:

$$\big\|\mathbf{r}_p - \mathbf{r}_j\big\| \leq h$$

Kelager gives the following equations to find the bounding box, by getting its two further corner points[13].

$$BB_{min} = \hat{\mathrm{x}}\big(\mathbf{r}_p - (h,h,h)^T\big)$$
$$BB_{max} = \hat{\mathrm{x}}(\mathbf{r}_p + (h,h,h)^T)$$

4.8 Collision Detection

For this implementation, the collision detection was handled very simply and only for a rectangular bounding box. The idea is that every time the particle hits a wall of the boundary, it “bounces” back, by having its velocity reversed:

$$\mathbf{u}_i = -\mathbf{u}_i$$

This method caused problems, especially in cases where the fluid forces were large, or where the fluid was thrown from a distance into the bounding box. In those cases, fully reversing the velocity made the particles jump up and down uncontrollably. The forces acting upon them could regulate the velocities, but there was not enough time for that before the particles hit another wall and their velocities were reversed once again.

For this reason, an energy loss parameter *s* is introduced, in order to decrease the velocity once the particle hits a wall. This is based on the fact that the particle would lose energy when colliding with the boundary and therefore would lose some of its velocity. The equation used for this new approach is:

$$\mathbf{u}_i = -s\mathbf{u}_i, \qquad s \geq 0$$

This approach, however, proved again problematic, because after continuously multiplying with the *s* value, the particles had more and more decreasing velocities. Because of this decrease, the velocities reach such small values that they end up almost “sticking” to the boundary walls and creating clusters near them, leaving the centre of the boundary very sparsely populated.

After encountering these problems, the *s* parameter was only used when the particles collided with the floor, in order to make them more “grounded” to it and not having them bounce straight off and fly away to the ceiling, while keeping the velocity intact when the particles hit any of the other boundary walls.

4.9 Physical Parameters

The physical parameters used in this implementation are:

- Mass: 0.02
- Volume: the volume is manually set, and for the simulations it varied from 1.5 to 6 m^3
- Particle size: 0.035 m^3
- Number of particles: the number depends on the volume and the particle size and is calculated from the following: $n = V/4\pi particle\_size^3$
- Smoothing kernel radius: varied from 0.15 to 1
- Rest density: 998.2 for water
- K constant: 5, used for the pressure
- Viscosity μ: 3.5
- Surface tension σ: 0.0728
- Surface tension threshold *l*: 6
- Buoyancy constant *b*: 0, for water

The viscosity and surface tension constants are the same that Kelager (2006) uses for his water simulation. He also proposes a way of finding the kernel size based on the volume, the particle count and the maximum amount of neighbours for a particle[13]:

$$max\_neighbours = \left(\frac{total_{particles}}{V}\right)\frac{4}{3}\pi h^3$$

$$\Rightarrow h = \sqrt[3]{\frac{3\,V\,max\_neighbours}{4\,\pi\,total\_particles}}$$

This equation however made the kernel size too small, approximately 0.15 which is too small of a number to get a good enough simulation. So the kernel size always has a size more than 0.25.

4.10 Classes and Pseudocode

The implementation of the fluid simulation was done in C++ using Qt[21] and the OpenGL[3] and ngl[4] libraries. For the purpose of that, 5 classes were created:

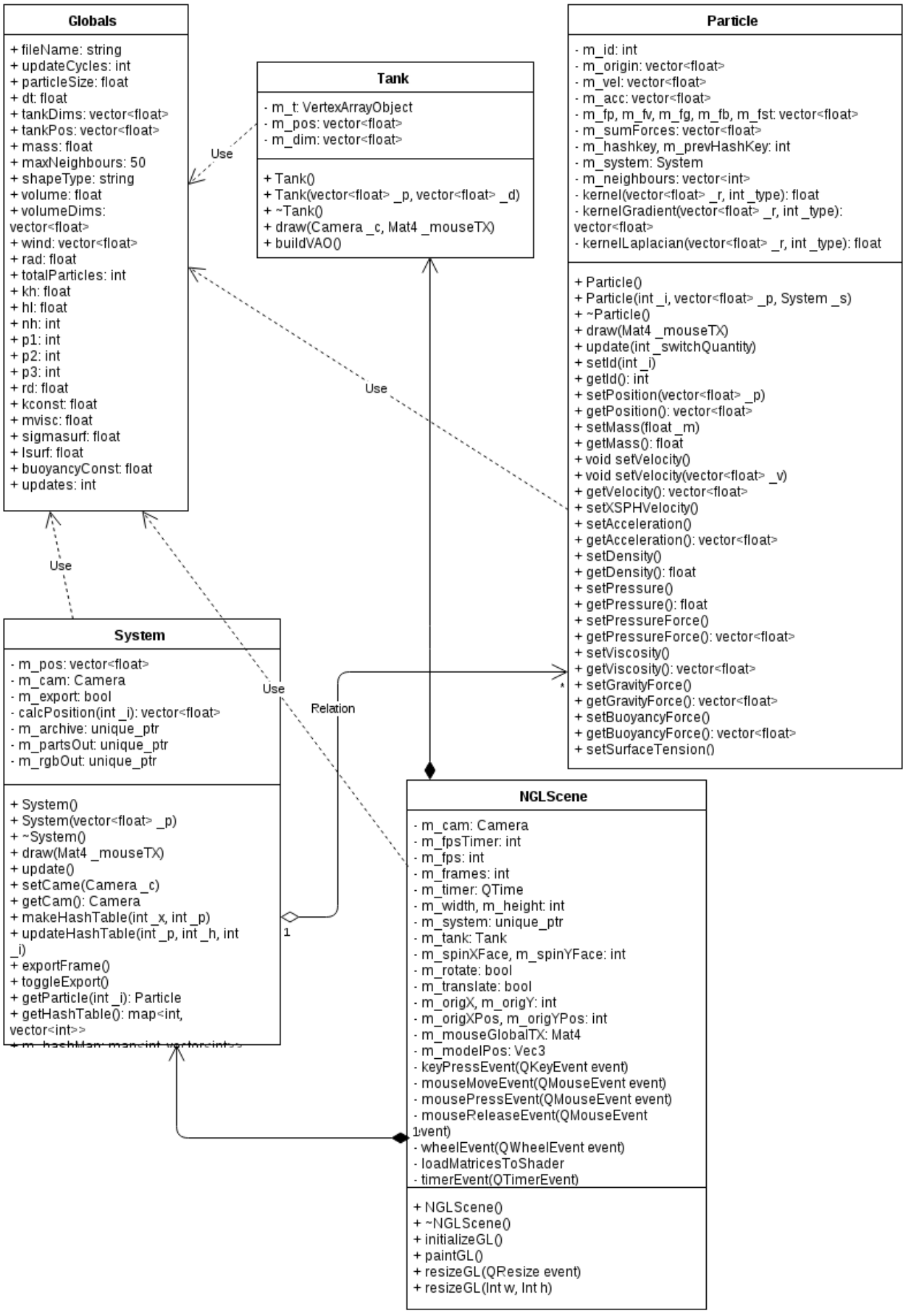
- *Globals*
- *NGLScene*
- *System*
- *Particle*
- *Tank*

Globals is a class where all the global variables used in the simulation are defined. The class is included in all the other classes so the globals are usable in them. *NGLScene* is the class that creates the scene, sets up and compiles the shaders and creates the camera. Part of the class was taken from Macey's[22] Simple Particles demo. The *System* class, that creates the particle system, and the Tank class are initialized inside *NGLScene*. Mouse and keyboard events are also handed through the class. *System* is the class that creates the particles, updates them and stores the particles inside a vector. It also stores the table needed for the spatial hashing, and contains functions for updating the it. *Particle* stores all the particle data and contains functions that perform all the calculations. Most of the main steps of the simulation go through this class. *Tank* is a class aimed to visualize the boundary box surrounding the fluid, by building a vertex array object.

The implementation is shortly described in the following pseudocode:

```
Set initial parameters and physical properties
Create system
Calculate size of hash table
For every particle p :
    Create p inside the initial volume
    Set p's mass
    Compute p's hash key
    Add p's id to the hash table
    Add particle p to the system's vector of particles
For every particle p :
    Set neighbours list
For every particle p :
    Calculate density
    Calculate pressure
For every particle p :
    Calculate internal forces
    Calculate external forces
    Sum up all forces
For every particle p :
    Update acceleration
    Update velocity
    Check boundary conditions and re-update velocity
While simulation running :
    For every particle p :
        Update position
        Compute new hash key and update hash table
        Set new neighbours list
    For every particle p :
        Calculate density
        Calculate pressure
    For every particle p :
        Calculate internal forces
        Calculate external forces
        Sum up all forces
    For every particle p :
        Update acceleration
        Update velocity
        Correct velocity using XSPH
        Check boundary conditions and re-update velocity
    For every particle p :
        Render p on screen
```

4.11 UML Diagram

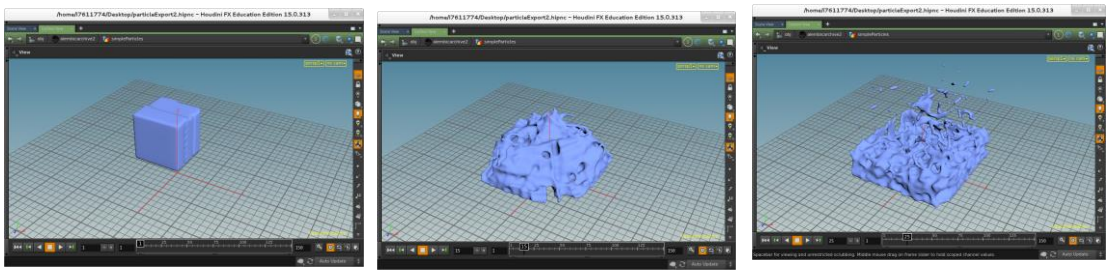


Chapter 5

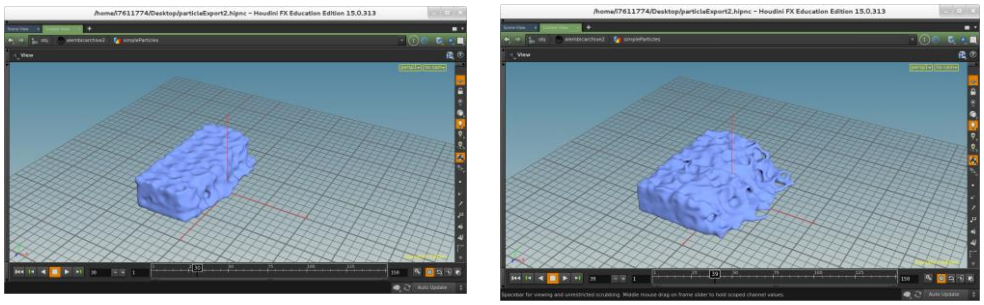
Results

Despite the multiple parameters and calculations, the factor that proved to be the most important for the simulation is the smoothing kernel size. The quality of the simulation depended largely on that parameter. Variations on the pressure and viscosity parameters were also tested. Here are screenshots from some of the simulations.

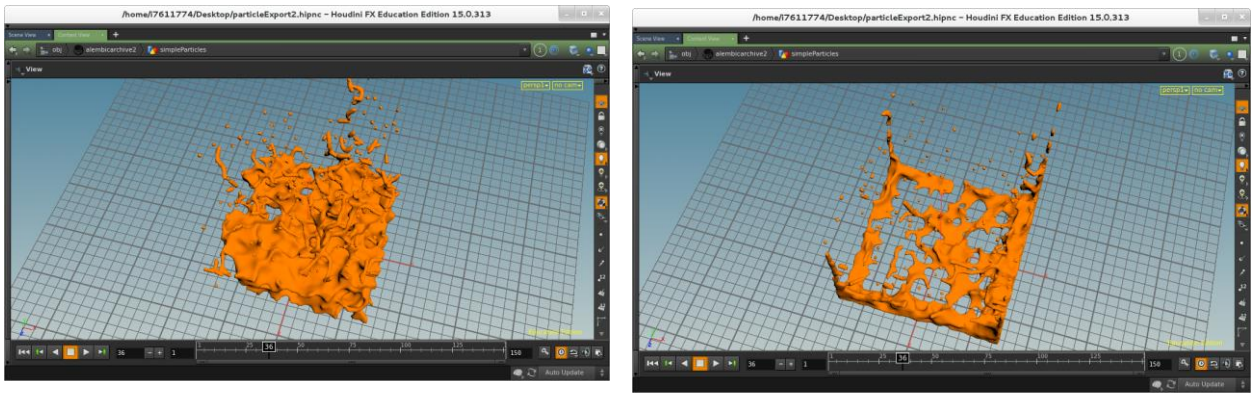
A falling cube:



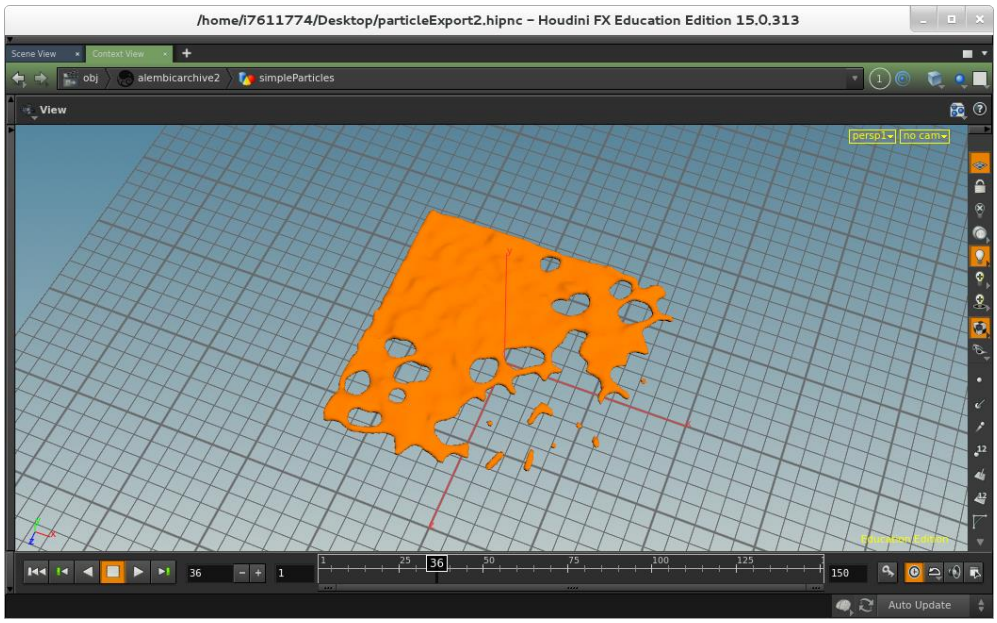
Dam break:

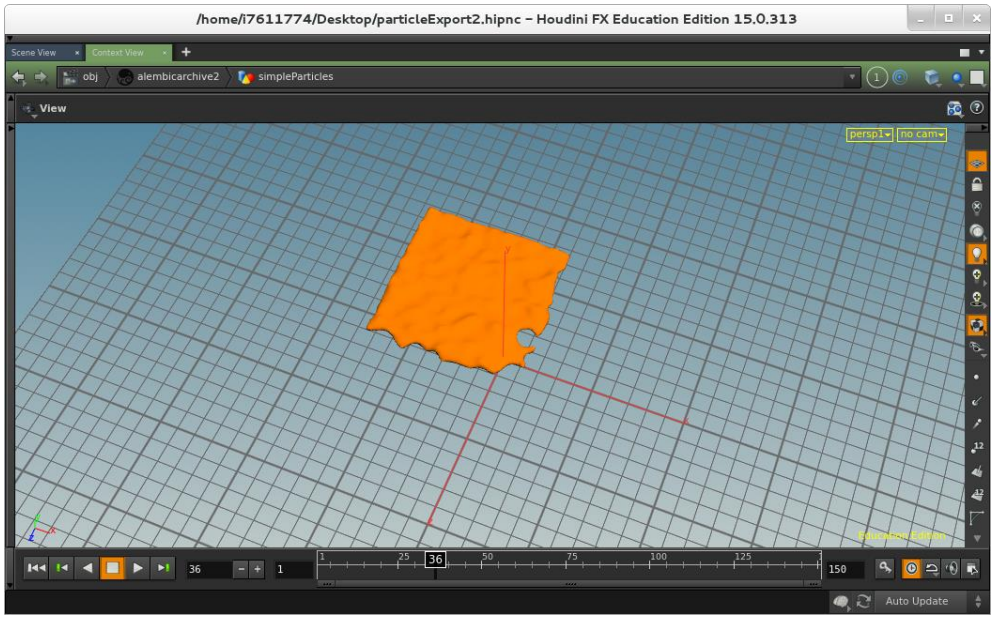


Variations on the viscosity parameter:



Variations on the kernel size, low size (above), high size (below):





Chapter 6

Conclusion

Smoothed particle hydrodynamics is a really powerful technique to simulate fluids and other similar phenomena. Part of its advantage is that the mathematics behind it, are not particularly complicated, as it happens with other fluid simulation techniques, like grid-based ones.

After this project, I have come to the conclusion, that the key to a good simulation using SPH, is the parameters. It is vital to find the right values and the right balance between the parameters as well, in order to have a simulation as realistic as possible.

6.1 Known Issues

6.1.1 Time performance

The biggest issue with this implementation was time. The simulation was running too slowly. As the amount of particles increased, the time needed to do all the calculations increased dramatically. The simulation became even slower when the smoothing kernel size increased. With a larger kernel size, each particle had to iterate over more neighbours during the calculations.

The fact that it was slow, made it also difficult to visualize. The simulation works best when it has several thousands of particles, above 7000-8000. But when the amount of particles exceeded 3000, the simulation became quite slow. Therefore, not many “test simulations” with 7000, or more, particles were done, in order to check which parameters make the simulation work better.

For example, in earlier versions, a simulation of 5000 particles, a kernel size as little as 0.2 would end up running for approximately 3 hours, in order to produce about 60 frames. Some optimization was done and the running time decreased a lot. One thing I did to optimize the code was change the way vector lengths were calculated. Vector lengths need a square root operation which is a costly one, so I tried to avoid that operation in cases where I could, like when comparing a vector length with another value. So instead of making this comparison:

$$\sqrt{\mathbf{x}_x{}^2 + \mathbf{x}_y{}^2 + \mathbf{x}_z{}^2} < a$$

where $\sqrt{\mathbf{x}_x{}^2 + \mathbf{x}_y{}^2 + \mathbf{x}_z{}^2} = \|\mathbf{x}\|$, we have this comparison:

$$\mathbf{x}_x{}^2 + \mathbf{x}_y{}^2 + \mathbf{x}_z{}^2 < a^2$$

which bypass the square root calculation, and has the same results, since we are talking about non-negative values. This step can save time from calculations, especially when calculating all the quantities involving kernel functions, which need distances between particles.

Another step that proved to optimize the code more effectively, was switching some class variables from private to public. Even though, in programming, it is considered better practice to not have class variables set to public, making that switch, made the simulation run faster. One particular variable that delayed the simulation a lot, was the std::map representing the hash table. This map was called inside the function that set neighbours for each particle. The table, which is a member of the System class, was accessed in the Particle class and through a get function. Since the neighbour setting class includes a lot of loops, the get function that retrieves the hash table could be called dozens of times, eventually delaying the simulation by a significant amount of time.

After these steps were taken, the aforementioned simulation example took about 10 minutes to run. When raising the smoothing kernel size from 0.2 to 0.35, it took about 25 minutes, while also raising the particle count from 5000 to 8500 made the simulation run for a little less than 1 hour, to produce 60 frames.

However, there was not enough time to optimize it further, and still some simulation ran for an overly long time. For example, a 100 frame simulation with 8500 particles and a kernel size of 0.45, took approximately 7 hours to run, whereas a simulation of 120 frames, 8500 frames but a kernel size of 0.35 took almost 3 and a half hours, which shows the vast difference that the kernel size can make time-wise.

6.1.2 Fluid restness

Besides the time problems, another problem in the simulation is that the fluid doesn't rest, after reaching the boundary box floor. While it should experience some sort of turmoil after landing properly on the floor, after a while, as long as there are no external forces besides gravity, that turmoil should cease. In some cases, after several frames have passed, the fluid stuck to the side of the boundary, probably because one of the forces is acting on it more than it should.

Trying to solve this issue, I experimented with two parameters, the kernel size and the *k* constant used for the pressure calculation. Too much pressure or too few neighbours made the fluid very "restless". Lowering the pressure constant made the fluid not flow too vigorously but didn't solve the problem completely. A larger kernel size had the particles not separate that easily, but, as mentioned in the above section, made the simulation very slow.

6.1.3 Volume changes

In the demo videos it can be seen that, before the fluid starts flowing into the tank, its volume starts shrinking down. This happens because there are not enough particles to fill the volume adequately, therefore they pull each other together, before their pressure forces start acting and they burst and flow into the tank.

6.1.4 Collision detection

As mentioned in chapter 4.8, collision detection was not ideal in the implementation. Decreasing the velocity when the particles were colliding with the boundary, made them "stick" to the wall after a few iterations, while not decreasing the velocity, but fully reversing it, made the particles move in an uncontrollable fashion.

6.2 Further Development

The implementation code has a lot of room for improvement. The first thing I would like to look into further, is try to optimize the code so that it runs fairly quickly. I would try to do that mostly by making the neighbour search more effective, since it is the most time consuming process, and also a critical one for the effectiveness of the simulation.

I would also like to look further into the particles' collisions with objects, so that the fluid interacts with spherical objects or other structures and not just a rectangular one.

References

[1] Englesson, D., Kilby, J. and Ek, J., 2011. *Fluid Simulation Using Implicit Particles*.

[2] Braley, C. and Sandu, A., 2010. *Fluid Simulation For Computer Graphics: A Tutorial in Grid Based and Particle Based Methods*. Blacksburg, Virginia, USA: Virginia Polytechnic Institute and State University.

[3] Khronos Group, 06 August 2012. *OpenGL*. 4.3 [3D graphics API]. Oregon, USA: Khronos Group.

[4] Macey, J., 2016. *NGL*. 6.5 [graphics library]. Bournemouth, UK: National Centre for Computer Animation, Bournemouth University.

[5] Lucasfilm Ltd., Sony Pictures Imageworks Inc., 2016. *Alembic*. [file format]. San Fransisco, California, USA: Lucasfilm Ltd., Vancouver, Canada: Sony Pictures Imageworks Inc..

[6] Side Effects Software, 2016. *Houdini*. 15.5 [computer program]. Toronto, Canada: Side Effects Software.

[7] Monaghan, J. J., 2005. *Smoothed Particle Hydrodynamics*. Bristol, UK: IOP Publishing Ltd. Available from: http://cg.informatik.uni-freiburg.de/intern/seminar/particleFluids_Monaghan%20-%20SPH%20-%202005.pdf.

[8] Gingold, R. A. and Monaghan, J. J., 1977. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181 (3), 375–389. Available from: http://mnras.oxfordjournals.org/content/181/3/375.full.pdf

[9] Lucy, L. B., 1977. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 82 (12), 1013–1024. Available from: http://articles.adsabs.harvard.edu/cgi-bin/nph-iarticle_query?1977AJ.....82.1013L&data_type=PDF_HIGH&whole_paper=YES&type=PRINTER&filetype=.pdf.

[10] Cline, D., Cardon, D. and Egbert, P. K., 2013. *Fluid Flow for the Rest of Us: Tutorial of the Marker and Cell Method in Computer Graphics*. Provo, Utah, USA: Brigham Young University.

[11] Stam, J., 1999. Stable Fluids. *SIGGRAPH '99 Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 121–128. Available from: http://dl.acm.org/citation.cfm?id=311548.

[12] Dobek, S., 2012. *Fluid Dynamics and the Navier-Stokes Equation (CMSC498A: Spring '12 Semester)*. College Park, Maryland, USA: University of Maryland, College Park.

[13] Kelager, M., 2006. *Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics*. Graduate Project. Department of Computer Science, University of Copenhagen.

[14] Priscott, C., 2010. *3D Lagrangian Fluid Solver using SPH approximations*. Masters Project. Bournemouth University.

[15] Becker, M. and Teschner, M., 2007. Weakly compressible SPH for free surface flows. *SCA '07 Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 209–217. Available from: http://dl.acm.org/citation.cfm?id=1272719.

[16] Müller, M., Charypar, D. and Gross, M., 2003. Particle-Based Fluid Simulation for Interactive Applications. *SCA '03 Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 154–159. Available from: http://dl.acm.org/citation.cfm?id=846298 [Accessed 21 August 2016].

[17] Harlow, F. H. and Welch, J. E., 1965. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. *The Physics of Fluids*, 8 (12), 2182–2189. Available from: http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S09/papers/harlow_welch.pdf

[18] Desbrun, M. and Gascuel, M.-P., 1996. Smoothed Particles: A new paradigm for animating highly deformable bodies. *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, 61–76. Available from: http://dl.acm.org/citation.cfm?id=274981.

[19] Batty, C., 2011. *Grid-Based Fluids*. British Columbia, Canada: The University of British Columbia. Available from: http://www.cs.ubc.ca/~batty/teaching/COMS6998/GridFluids_overview.pdf

[20] Teschner, M., Heidelberger, B., Müller, M., Pomeranets, D. and Gross, M., 2003. Optimized Spatial Hashing for Collision Detection of Deformable Objects. *Proceedings of Vision, Modeling, Visualization VMV'03*, 47–54. Available from: http://matthias-mueller-fischer.ch/publications/tetraederCollision.pdf.

[21] The Qt Company, 01 July 2015. *Qt*. 5.5 [application framework]. Espoo, Finland: The Qt Company.

[22] Macey, J., 2016, *Alembic Export*. [coding demo]. Bournemouth, UK: National Centre for Computer Animation, Bournemouth University.

[23] Harlow, F. H., 1962. *Computer Physics Communications*. Los Alamos, New Mexico: The University of California, Los Alamos Scientific Laboratory.

[24] Bridson, R., 2015. *Fluid Simulation for Computer Graphics*. 2nd edition. Boca Raton, Florida, USA: CRC Press.

[25] fxguide, 2011. *The science of Fluid Sims*. fxguide. Available from: https://www.fxguide.com/featured/the-science-of-fluid-sims/ [Accessed 22 August 2016].

[26] Harlow, F. H., 2004. Fluid dynamics in group T-3 Los Alamos national laboratory. *Journal of Computational Physics*, 195 (2), 414–433. Available from: http://dl.acm.org/citation.cfm?id=992975.

[27] Monaghan, J. J., 1992. Smoothed particle Hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30 (1), 543–574. Available from: http://www.astro.lu.se/~david/teaching/SPH/notes/annurev.aa.30.090192.pdf.

[28] Reeves, W.T., 1983. Particle Systems – A Technique for Modeling a Class of Fuzzy Objects [online]. *ACM Transactions on Graphics (TOG)*, Volume 2 Issue 2, Pages 91-108.

[29] *Star Trek II: The Wrath of Khan*, 1982. [film, DVD]. Directed by Nicholas Meyer. USA: Paramount Pictures.

[30] Siggraph, 2000. *Particle Systems* [online]. Available from: https://www.siggraph.org/education/materials/HyperGraph/animation/particle.htm [Accessed 15 August 2016]

[31] Brackbill, J. U. and Ruppel, H. M., 1986. FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *Journal of Computational Physics*, 65 (2), 314–343. Available from: http://dl.acm.org/citation.cfm?id=9229 [Accessed 21 August 2016].