Divergence-Free Smoothed Particle Hydrodynamics
Advanced Simulation and Visualization of Fluids in Computer Graphics

Isabell Jansson[*] Ronja Grosz[†] Jonathan Bosson[‡]

January 16, 2017

Abstract

This paper discusses the technique presented by Bender et. al. [1] for simulating incompressible fluids with an efficient and stable Divergence-Free Smoothed Particle Hydrodynamics method. The method uses two solvers to obtain a divergence-free fluid, a divergence solver and a density solver to correct the density after the divergence has been corrected. This results in a stable and fast simulation of a divergence-free fluid even though the simulation is performed on the CPU without any parallelization.

Index Terms: Incompressible fluids, Divergence-free, SPH, divergence correction, density correction.

1 Introduction

Smoothed particle hydrodynamics, *SPH*, is a method that was first implemented in 1977 for astrophysical simulations by Gingold et al. [2]. Since then SPH has become a popular method for complex fluid simulations. SPH is a mesh-free Lagrangian method where the fluid is split into discrete sets defined as particles, which move in space and change physical properties as time progresses.

In this paper we are going to introduce the results from reproducing the divergence-free smoothed particle hydrodynamics method presented by Bender et al. [1]. It is a method which corrects the divergence error, aiming for a divergence-free velocity field which is needed for an incompressible fluid. For the solution to be divergence-free the density has to be constant over time.

2 Background and Related Work

In the field of computer graphics a variety of methods for simulating incompressible fluids exist. This section will give a brief overview of related approaches.

Bender et al. [1] proposed a method for a stable implicit SPH solution. The method uses a combination of two pressure solvers which enforce a low volume compression below 0.01% and a divergence-free velocity field. It can be seen as enforcing incompressibility on both position and velocity level. The low compression is important for realistic physical behavior and a divergence-free state increases the stability of the simulation which reduces the number of solver iterations.

Another incompressible SPH method was proposed by Hu et al. [3] for multiphase flows where the time integration step to obtain velocities is divided into two half steps. This is done to enforce both zero density variation condition and the velocity divergence-free condition at each full time step. During the first half step density fluctuations are

[*]isaja187@student.liu.se
[†]rongr946@student.liu.se
[‡]jonbo665@student.liu.se

eliminated by altering the intermediate particle position. During the second half step errors in velocity divergence are resolved by altering the intermediate particle velocity.

Ihmsen et al. [4] proposed an implicit incompressible SPH method with a pressure Poisson equation made by the combination of a symmetric SPH pressure force and a SPH discretization of the continuity equation. The pressure Poisson equation is used to compute the pressure so that the pressure forces correct the intermediate velocities to a divergence-free state.

A predictive corrective incompressible SPH method was proposed by Solenthaler et al. [5] based on the Lagrangian SPH model. Incompressibility was enforced using a prediction correction scheme to determine the particle pressures. To achieve this the information about the density fluctuation was actively propagated through the fluid and the pressure values were updated until the targeted density was satisfied.

A ghost fluid approach for free surface and solid boundary conditions for a SPH fluid simulation was proposed by Schechter et al. [6]. The ghost particles are placed on free surfaces and at solid boundary conditions. The approach solves problems like spurious numerical surface tension artifacts and errors in the mass conservation constraints.

3 Method

The produced simulation after following the method proposed by Bender et al. [1] is drafted in Listing 1. Upon starting the application the particle neighbourhood N_i will be determined with a cell list solution explained in Section 3.3. For each particle the alpha α_i and density ρ_i parameters will be computed, both of which are commonly reused variables and thus worth storing to reduce computation cost. This is done before entering the simulation loop as initial values are required. During the loop all these parameters will be updated as the simulation progresses.

Once in the simulation at line 7 the first time step will be determined through the CFL condition explained in Section 3.2. The new velocities for each particle is then predicted through the included external forces which are propagated with ordinary Euler integration.

```
1  function performFluidSimulation
2    for all particles $i$ do //Init neighbourhoods
3      find neighbourhoods $N_i(0)$
4    for all particles $i$ do //Init $\rho_i$ and $\alpha_i$
5      compute densities $\rho_i(0)$
6      compute factors $\alpha_i(0)$
7    while (t < $t_{max}$) do //Start simulation loop
8      adapt time step $\Delta t$
9      for all particles $i$ do //Predict velocities $\mathbf{v}_i^*$
10       $\mathbf{v}_i = \mathbf{v}_i + \Delta t \mathbf{F}_g / m_i$
11     correctDensityError($\alpha, \mathbf{v}_i$) // Fulfill $\rho^* - \rho_0 = 0$
12     for all particles $i$ do //Update positions
13       $x_i(t + \Delta t) = x_i(t) + \Delta t \mathbf{v}_i^*$
14     for all particles $i$ do //Update neighbourhoods
15       find neighbourhoods $N_i(t + \Delta t)$
16     for all particles $i$ do //Update $\rho_i$ and $\alpha_i$
17       compute densities $\rho_i(0)$
18       compute factors $\alpha_i(0)$
19     correctDivergenceError($\alpha, \mathbf{v}^*$) // Fulfill $\frac{D\rho}{Dt} = 0$
```

Listing 1: Simulation algorithm

The density solver uses the precomputed parameter α_i together with this prediction to compute the pressure forces in each neighbourhood such that it can correct the density error $\rho_i^* - \rho_i = 0$. This is further explained in Section 3.6.

After the particle position $\boldsymbol{x_i}$ has been updated the neighbourhoods and the α_i and ρ_i factors are recalculated to reflect the new positions. Lastly, in line 19 of Listing 1 the divergence solver, much like the density solver, calculates the pressure forces to eliminate its error. In this case the divergence error $\frac{D\rho_i}{Dt}$ is corrected to be equal to 0.

3.1 Navier-Stokes

Navier-Stokes equations describe incompressible fluids and are therefore suitable for SPH simulations. The equations expressed in Lagrangian coordinates can be found in Equation 1 and 2.

$$\frac{D\rho}{Dt} = 0 \Leftrightarrow \nabla \cdot \mathbf{v} = 0 \tag{1}$$

$$\frac{D\mathbf{v}}{Dt} = -\frac{1}{\rho}\nabla p + v\nabla^2\mathbf{v} + \frac{\mathbf{f}}{\rho} \tag{2}$$

In the equations $\frac{D\rho}{Dt}$ and $\frac{D\mathbf{v}}{Dt}$ denotes the derivate of the density ρ and the velocity $\mathbf{v}$ respectively. The pressure, kinematic viscosity and body forces are denoted by p, v and $\mathbf{f}$ respectively. The second part of Equation 1, $\nabla \cdot \mathbf{v} = 0$, induce that the velocity field is free from divergence. From the continuity equation $\frac{D\rho}{Dt} = -\rho\nabla \cdot \mathbf{v}$ and the divergence-free condition it follows that the partial derivate of the density with respect to the time is zero, which implies the equivalence of the equations in Equation 1. From this it follows that the density, in theory, must stay constant over time which is the implication of an incompressible fluid. In practice the divergence-free condition is not sufficient to guarantee incompressibility in simulations. The numerical time integrations will contain numerical errors which will cause density deviations due to the volume compressions. To avoid this a second condition $\rho - \rho_0 = 0$ called *constant density condition* must be fulfilled [1]. Section 3.6 and 3.7 give a deeper description of the solvers used in the project.

3.2 Adapted time step

In order to assure a stable simulation it is necessary to adapt the time step depending on the motion. Depending on how quick the particles move, smaller time steps are taken such that the numerical error of the explicit time integration is negligible. A condition was proposed by Courant−Friedrichs−Lewy, often refered to as the CFL-condition, which is shown in Equation 3. The time step Δt is equal to a fraction of the particle's diameter d divided by the maximum velocity $\boldsymbol{V}_{max}$ in the scene.

$$\Delta t \leq \frac{0.4 \cdot d}{\boldsymbol{V}_{max}} \tag{3}$$

Each rendered frame however have a constant time step in order to ensure no speed change in the playback of the rendered video. This is decided from the user side as a desired fps parameter. This means that the simulation loop can iterate multiple times within each frame adaptively depending on the maximum velocity of a particle.

3.3 Neighbourhood search

Since SPH only considers a finite amount of neighbouring particles it is important to keep track of every particles' neighbours. Searching through all particles for neighbours within the cutoff distance h for every particle is inefficient and takes $\mathcal{O}(N^2)$ time. The cutoff distance h is the kernels' smoothing radius. To make this faster a cell list was implemented. A cell list is a data structure that is divided into cells that have a length larger or equals to the cutoff distance h. Each cell spans on a surface in space. A particle belongs to a cell if it is inside the space the cell occupy. When finding the neighbour of particle i, only the neighbouring cells have to be searched for particles within the cutoff distance, see Figure 1.

The cell list is implemented by using a four dimensional vector where the first three dimensions are for the x, y and z coordinates and the fourth dimension is for storing the index of the particles that belong to the cell.

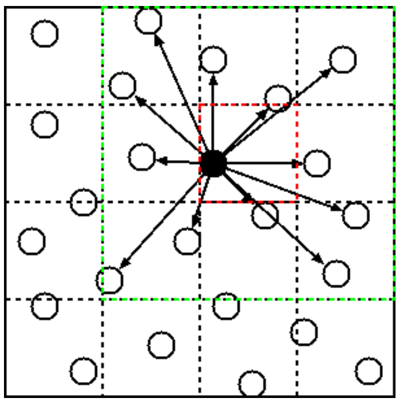

Figure 1: Finding the neighbours for the filled in particle i by looking through all neighbouring cells, including its own cell, for particles within the cutoff distance h

The amount of cells are decided by dividing the scene into cells of length h. The particles are then assigned to a cell by finding the cell's coordinates that it belongs to according to Equation 4, where $\boldsymbol{x}$ is the position of the particle and $\boldsymbol{s}_{min}$ is the lowest position of the scene space. If the particle moves out of its cell it is reassigned to the new cell it is inside.

$$\frac{\lceil \boldsymbol{x} - \boldsymbol{s}_{min} \rceil}{h} \tag{4}$$

3.4 Kernel

A kernel function is used to simulate how particle to particle interactions decrease with the distance between the current particle and its neighbours. In SPH simulations this is an approximation of the Gaussian kernel function. Different kernels have been tested in previous works i.e. the poly6 kernel, the spiky kernel and the cubic spline kernel. According to Bender et. al. [1] the cubic spline kernel presented by Monaghan [7] was used. The kernel is described by Equation 5, where $q(x) = \frac{\|x\|}{h}$, x is the distance between the current particle and a neighbour particle and h is the support radius for the kernel. Particles further away than the support radius will not affect the current particle.

$$W_h(q(x)) = \frac{1}{\pi h^3} \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3 & 0 \leqslant q < 1 \\ \frac{1}{4}(2-q)^3 & 1 \leqslant q < 2 \\ 0 & q \geqslant 2 \end{cases} \tag{5}$$

The algorithm does also require the kernel gradient. To reduce the computational effort and memory requirements Bender et. al. [1] introduce a scalar function $g(q) = \frac{\partial W_h}{\partial q} \cdot \frac{1}{h\|x\|}$. The gradient kernel is then calculated by $\partial W_h(q(x)) = x \cdot g(x)$. The gradient kernel is described by Equation 6.

$$\partial W_h(q(x)) = x \frac{1}{h\|x\|} \frac{1}{\pi h^3} \begin{cases} -3q + \frac{9}{4}q^2 & 0 \leqslant q < 1 \\ -\frac{3}{4}(2-q)^2 & 1 \leqslant q < 2 \\ 0 & q \geqslant 2 \end{cases} \tag{6}$$

It is important to use the same kernel function for both W_h and ∇W_h to get the prediction and the correction step to be compatible to each other.

3.5 Density and alpha factors

The density in a region of the fluid with a particle x_i in the center is calculated through Equation 7 where ρ_i is the density with the current particle x_i in center, ρ_j is the density with a neighbouring particle x_j in center, m_j is the mass for a neighbour particle and W_{ij} is the kernel function described in Section 3.4. This implies that neighbouring particles closer to the current particle x_i will have a greater affect on the density.

$$\rho_i = \sum_j \frac{m_j}{\rho_j} \rho_j W_{ij} = \sum_j m_j W_{ij} \tag{7}$$

Both the density solver and the divergence solver uses a stiffness parameter κ_i^v presented by Bender et. al. [1], Equation 8.

$$\kappa_i^v = \frac{1}{\Delta t}\frac{D\rho_i}{Dt} \cdot \alpha_i \tag{8}$$

In the equation Δt denotes a small time step and α_i is a precomputed factor to decrease the computations while performing the solvers. Bender et. al. [1] defines α_i as

$$\alpha_i = \frac{\rho_i}{\left|\sum_j m_j \nabla \boldsymbol{W}_{ij}\right|^2 + \sum_j \left|m_j \nabla \boldsymbol{W}_{ij}\right|^2} \tag{9}$$

where $\boldsymbol{W}_{ij}$ corresponds to the kernel gradient.

3.6 Density solver

This solver aims to minimize the density error caused by the numerical integration in the divergence solver. The density error is determined by comparing the actual density to the rest density, $\rho - \rho_0$. There are several density solvers presented in previous works, but the solver used in this project was presented by Bender et. al. [1] which uses the fact that α_i has been calculated before the solver which reduces the computations in the solver. The solver will be active while the average of the predicted densities for every particle differs from the rest density with an error greater than a constant *maxError*. For every particle a predicted density ρ_i^* is calculated through an Euler integration step by Equation 10.

$$\begin{gathered}\rho_i^* = \rho_i + \Delta t \tfrac{D\rho_i}{Dt} \\ \Leftrightarrow \\ \rho_i^* = \rho_i + \Delta t \textstyle\sum_j m_j(\mathbf{v}_i^* - \mathbf{v}_j^*)\nabla \boldsymbol{W}_{ij}\end{gathered} \tag{10}$$

$$\boldsymbol{F_i^p} = -\frac{m_i}{\rho_i}\nabla \boldsymbol{p_i} \tag{11}$$

The pressure force of a particle is determined by Equation 11, where the pressure gradient can be solved using the SPH formulation proposed by Ihmsen et al. [8], Equation 12.

$$\nabla \boldsymbol{p_i} = \kappa_i^v \nabla \rho_i = \kappa_i^v \sum_j m_j \nabla \boldsymbol{W_{ij}} \tag{12}$$

$$\boldsymbol{F^p_{j\leftarrow i}} = -\frac{m_i}{\rho_i}\kappa_i^v m_j \nabla \boldsymbol{W}_{ij} \tag{13}$$

Further on, the pressure forces $\boldsymbol{F^p_{j\leftarrow i}}$ that act from particle i on all its neighbours j is calculated through Equation 13, where the stiffness parameter κ_i is determined by Equation 14.

$$\kappa_i = \frac{1}{\Delta t^2}(\rho_i^* - \rho_0)\alpha_i \tag{14}$$

To conserve momentum it is required that all inner pressure forces sum up to zero such that $\boldsymbol{F_i^p} + \boldsymbol{F^p_{j\leftarrow i}} = 0$. As proposed by Bender et. al [1] the solver computes the total force $\boldsymbol{F^p_{i,total}}$ through Equation 15.

$$\begin{gathered}\boldsymbol{F^p_{i,total}} = \boldsymbol{F_i^p} + \textstyle\sum_j \boldsymbol{F^p_{i\leftarrow j}} \\ \Leftrightarrow \\ \boldsymbol{F^p_{i,total}} = -m_i \textstyle\sum_j m_j(\frac{\kappa_i^v}{\rho_i} + \frac{\kappa_j^v}{\rho_i})\nabla \boldsymbol{W}_{ij}\end{gathered} \tag{15}$$

Newton's second law $F = ma = m\frac{dv}{dt}$ implies that $\frac{dv}{dt}$ is according to Equation 16. The velocity of particle i is thus changed according to Equation 17.

$$\frac{dv}{dt} = \sum_j m_j(\frac{\kappa_i^v}{\rho_i} + \frac{\kappa_j^v}{\rho_i})\nabla \boldsymbol{W}_{ij} \tag{16}$$

$$\boldsymbol{v^*_{i+1}}(t) = \boldsymbol{v_i^*}(t) - \Delta t \textstyle\sum_j m_j(\frac{\kappa_i^v}{\rho_i} + \frac{\kappa_j^v}{\rho_i})\nabla \boldsymbol{W}_{ij} \tag{17}$$

3.7 Divergence solver

The divergence-free solver ensures the fluid to be free of density change over time by enforcing $\frac{D\rho}{Dt} = 0$, which is equivalent to the Navier-Stokes divergence free condition given in Equation 1.

Equation 18 defines how the density change is computed. As in the density solver, the total pressure forces acting on a particle i are computed. The stiffness parameter is calculated through Equation 19.

$$\frac{D\rho_i}{Dt} = \sum_j m_j(\boldsymbol{v_i} - \boldsymbol{v_j})\nabla \boldsymbol{W}_{ij} \tag{18}$$

$$\kappa_i^v = \frac{1}{\Delta t}\frac{D\rho_i}{Dt} \cdot \alpha_i \tag{19}$$

The stiffness parameters need to be determined iteratively since neighboring particles depend on each other. After the symmetric pressure force has been applied to the new velocity should the density change in Equation 18 be equal or close to zero. If the error is too large the process will be iterated until the desired result have been achieved.

3.8 Boundary conditions

Dirichlet boundary condition was implemented in order to simulate the fluid inside a bounding box. Dirichlet assumes that the pressure is zero outside the fluid and therefore the value a quantity has to take along the boundary has to be specified, in this case the velocity was set to zero when a particle reaches the boundary, Bridson [9].

An isosurface was also introduced in the middle of the scene in order to simulate collisions with arbitrary objects. This was done as an implicit representation of geometric shapes, described as a function $f(x, y, z)$. Given an isovalue $C = f(\boldsymbol{x})$, point $\boldsymbol{x}$ can be classified as follows:

$$\begin{array}{ll} \text{Inside:} & f(\mathbf{x}) < C \\ \text{Outside:} & f(\mathbf{x}) > C \\ \text{On surface:} & f(\mathbf{x}) = C \end{array} \tag{20}$$

All possible implicit geometries can be described by a quadratic function, which is found in Equation 21, and in matrix form in Equation 22.

$$\begin{aligned} f(x,y,z) &= Ax^2 + 2Bxy + 2Cxz \\ &+ 2Dx + Ey^2 + 2Fyz \\ &+ 2Gy + Hz^2 + 2Iz \\ &+ J \end{aligned} \tag{21}$$

$$\mathbf{p}^T\mathbf{Q}\mathbf{p} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} A & B & C & D \\ B & E & F & G \\ C & F & H & I \\ D & G & I & J \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{22}$$

Considering that the geometry is placed at the origin, $f(\boldsymbol{x}) = 0$ means the particle is on the surface and $\boldsymbol{p}^T\boldsymbol{Q}\boldsymbol{p} \leq 0$ can be used to determine collisions.

Since the quadric surface is known analytically the differentiation can be applied to the quadric directly to get the gradient of the shape in position $\boldsymbol{x}$. This gives an efficient and analytical expression with a coefficient matrix Q, which is used to help determine the new velocity direction of a particle after colliding with the surface.

$$\begin{aligned} \nabla f(x,y,z) = 2 \begin{bmatrix} A & B & C & D \\ B & E & F & G \\ C & F & H & I \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \\ = 2\mathbf{Q}_{sub}\mathbf{p} \end{aligned} \tag{23}$$

4 Implementation

The divergence-free SPH simulation was developed using OpenGL and C++ together with the OpenGL mathematics library GLM for vector calculations. FreeImage was used to save frames of the simulation and FFmpeg to assembled the frames into a video. PicoJSON was used to load variables from a file making it easier to change variables quickly and efficiently and a makefile was used to compile the project.

The simulation was developed on macOS and Linux. The Linux computer is running on Debian and has a 50 GHz Intel Core i5-2450M processor and a NVIDIA GeForce 610M graphics card. One of the Mac OS computer is running on macOS Sierra and has a 1.8 GHz Intel Core i5 processor with an Intel HD Graphics 400 1536 MB graphics card.

5 Results

When the application starts, the water is represented as set of spheres where every sphere has the properties of a particle. The initial state is illustrated in Figure 2. The water is affected by a gravity which induce a falling motion, illustrated in Figure 3.

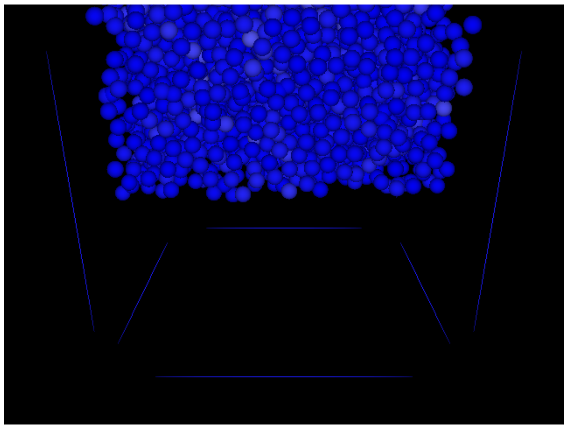
Figure 2: Initial state of the water when the application starts.

Since the water aims to be in the state of rest, the motion will decrease over time which is illustrated in Figure 4.

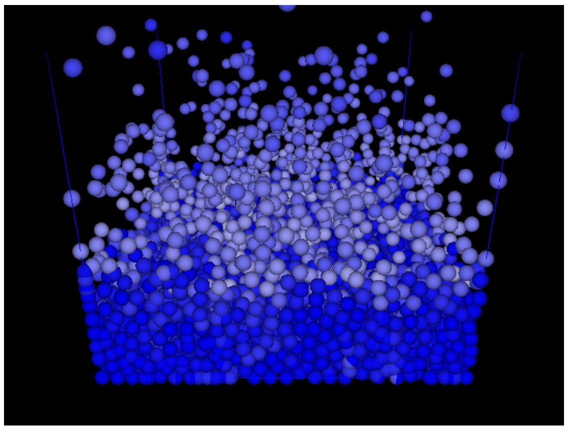
Figure 3: The water falls due to the gravity.

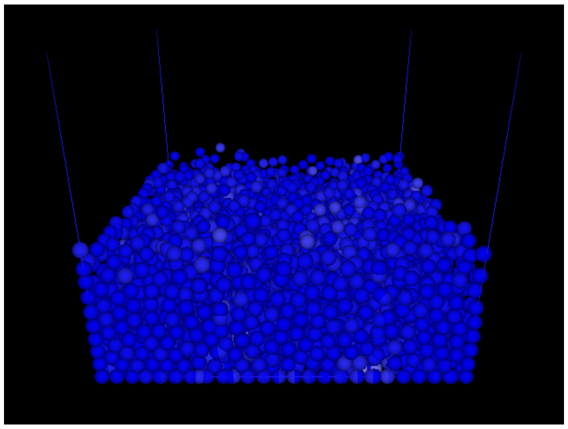
Figure 4: The water is calm with some motion.

To be able to simulate how outer forces affect the water, we implemented a feature where the user is able to change the direction of the force. The result when applying a force from right to left on the water and then letting only the gravity affect it is shown in Figure 5.

Implicit geometries can also be added to the scene in order to see how the water collides and interacts with it. Figure 6 illustrates a scene with a solid cylinder that forces the water to divert its momentum around the cylinder's surface.

When running the program in real time with 5000 particles it runs with a frame rate

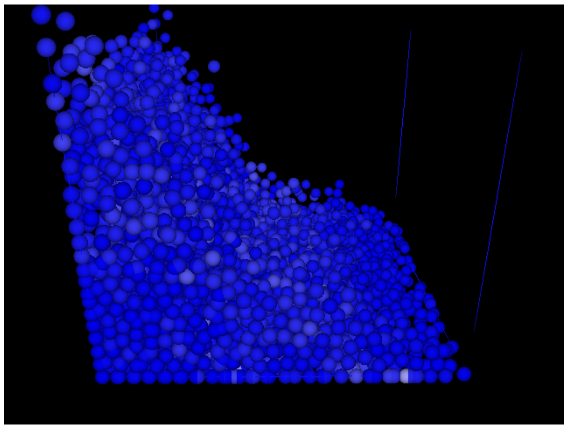

Figure 5: The water after an applied right to left motion followed by gravity as the only outer force.

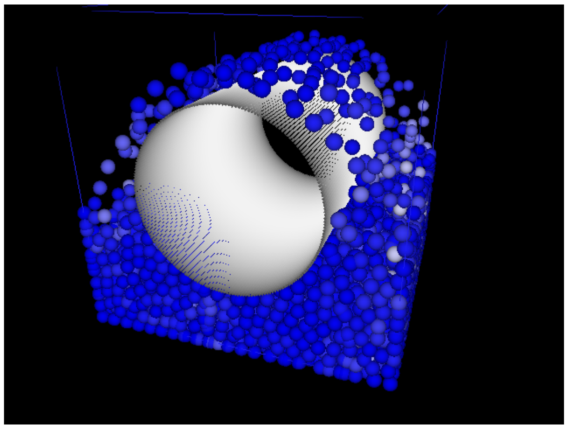

Figure 6: The water colliding with a cylinder.

of 6-8 fps on the MAC computer. When instead simulating with $10,000$ particles the frame rate is about 2 fps on the MAC computer.

6 Conclusions and Future Work

We were able to implement the SPH method presented by Bender et al. [1] which induce that the fluid is incompressible. Though, there are several possible improvements that would improve the implementation and increase the frame rate.

To increase the accuracy of the interaction with the boundary and free surfaces a method like ghost-SPH that was presented by Schechter et al. [6] could be implemented. A method that Bender et al. [1] used was the versatile rigid fluid coupling method for incompressible SPH presented by Akinci et al. [10]. It is based on static boundary particles which could also be a valid implementation to get better boundary interactions.

To get the simulation to be able to handle more particles the solution could be parallelized to speed up the calculations. This could be done using OpenMP and increasing the threading.

To get the simulation more interesting viscosity can be introduced into the fluid simulation. It would make simulations of thicker fluids possible.

According to Bender et al. [1] the kernel values could easily be precomputed and used together with a look up-table in order to speed up the calculations. This will result in a 30% faster application. This could be done once when the iteration loop begins and the positions for the particles are determined. Another approach would be to compute a discrete representation of the kernel function before the iteration loop and interpolate between to kernel values in the simulation loop in order to find a kernel value for a specific particle.

References

[1] J. Bender and D. Koschier, “Divergence-free smoothed particle hydrodynamics,” in *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 147–155, ACM, 2015.

[2] R. A. Gingold and J. J. Monaghan, “Smoothed particle hydrodynamics: theory and application to non-

spherical stars," *Monthly notices of the royal astronomical society*, vol. 181, no. 3, pp. 375–389, 1977.

[3] X. Hu and N. A. Adams, "An incompressible multi-phase sph method," *Journal of computational physics*, vol. 227, no. 1, pp. 264–278, 2007.

[4] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, and M. Teschner, "Implicit incompressible sph," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 3, pp. 426–435, 2014.

[5] B. Solenthaler and R. Pajarola, "Predictive-corrective incompressible sph," in *ACM transactions on graphics (TOG)*, vol. 28, p. 40, ACM, 2009.

[6] H. Schechter and R. Bridson, "Ghost sph for animating water," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 61, 2012.

[7] J. J. Monaghan, "Smoothed particle hydronamics," *Annual review of astronomy and astrophysics.(A93-25826 09-90), p.* ., vol. 30, pp. 543–574, 1992.

[8] M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, and M. Teschner, "Sph fluids in computer graphics," 2014.

[9] R. Bridson, *Fluid Simulation for Computer Graphics*. Ak Peters Series, Taylor & Francis, 2008.

[10] N. Akinci, M. Ihmsen, G. Akinci, B. Solenthaler, and M. Teschner, "Versatile rigid-fluid coupling for incompressible sph," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 62, 2012.