Volume 0 (1981), Number 0 pp. 1–6 COMPUTER GRAPHICS *forum*

GPU-Based SPH Fluid Simulation

J. Youngberg

University of Victoria

Abstract

Fluid simulations are very common in the field of computer animation and there are many different techniques for doing implementing them. Smoothed Particle Hydrodynamics (SPH) is one of the most common techniques. This approach simulates fluid as a set of particles moving through space and interacting with each other. There has a been a lot of research dedicated to optimizing and improving this method. Today, SPH is often used to simulate fluids in film and virtual environments. With the advancement of graphics cards, it is now preferred to run fluid simulations on the GPU, which massively improves performance through parallelization. This paper proposes a GPU-based SPH fluid simulation implementation based on a few noteworthy research papers. The results verify the research since an efficient SPH fluid simulation was successfully implemented.

1. Introduction

Fluids are everywhere in the world. Nautally, they are often required in film, games, or any other virtual environment. In order for them to appear real, they must be simulated based on the rules of fulid dynamics. Unfortunately, doing this in a realistic way is very computationaly expensive. For this reason, there has been a lof of research on how to simulate fluids efficiently, so that users can interact with them in real time. The research on simulating fluids can be classified into Eulerian, which is grid based, and Lagrangian, which is particle based.

In Eulerian methods, the computation happens for fixed points in space that density flows through. In Lagrangian methods, the computation happens for particles that move through space. Both methods have their strengths and weaknesses. While any implementation is based off one, it often incorporates ideas from the other. For example, Eulerian methods poorly handle advection, so they often use a semi-Lagrangian approach for that part of the simulation. Similarly, Lagrangian approaches require searching for particle neighbors, so they often discretize the space to narrow down possible neighbors.

A popular Lagrangian approach is Smoothed Particle Hydrodynamics (SPH). In this method, forces for each particle are calculated as weighted sums of their neighbors, smoothed using something called a kernel function. A kernel function takes the distance as input and returns a contribution amount for a given neighbor. Each particle is only affected by other particles that are within a certain distance called the kernel radius, so only a select few may affect a given particle's motion. As mentioned earlier the main problem with this approach is that searching for neighbors can be very expensive. Since only some particles may actually be within the kernel radius, the space can be divided into a grid to reduce search complexity. In this way, to find the neighbors of a particle, only neighboring bins of the given particle's bin need be searched to find the neighbors (9 bins in 2D, 27 in 3D). Still, to acheive interactive frame rates with many particles, the simulation must be performed on GPU, otherwise it is simply too expensive. Most modern implementations of SPH are parallelized on the GPU.

2. Related Work

Grid-based approaches have a long history of use for fluid simulation, dating back to Harlow developing the particle-in-cell method (PIC) in 1962 [Har62], and the marker-and-cell (MAC) methods with Welch in 1965 [HW65]. More recently, Foster and Metaxas introduced a Eulerian approach to solve the Navier Stokes equations with finite-differences [FM96]. Since then, grid-based methods have become very popular. Stam soon after improved Eulerian advection problems by introducing semi-Lagrangian adveection [Sta99]. Later, Fedkiw and Foster introduced the level set method to improve mass conservation [FF01]. Since then there has been many improvements to grid based methods such as improved boundary conditions by Rasmussen *et al.* [REN*04], surface tracking by Bargteil *et al.* [BGOS06], and multi-fluid interactions by Losasso *et al.* [LSSF06].

Lagrangian approaches use moving particles to simulate fluid motion. Two such approaches are Moving Particle Semi-Implicit (MPS) [KO96] and Smoothed Particle Hydrodynamics (SPH) [Mon92]. SPH was originally developed for fluid simulation in astronomy by Monaghan, and was later introduced to computer graphics by Desbrun and Gascuel [DG96]. Müller *et al.* used SPH to solve the Navier Stokes Equations, creating on of the first interactive fluid simulations using the Navier Stokes equations [MCG03]. Still, this simulation was limited to a few thousand

particles, not nearly enough to simulate larger bodies of water. With the advancement of GPUs, it has become possible to use it for non graphic tasks, like simulation or linear algebra. Thus, a lot of research has gone into accelerating fluid simulations with the GPU. While Eulerian approaches more easily lend themselves to parellization, it is hard to efficiently parallelize SPH. This is largely due to the neighbor search being a bottleneck, which is of time complexity $O(n^2)$ if all particles are searched. Early research into GPU-based SPH fluid simulations still suffered a great deal from this bottleneck. Harada *et al.* proposed one of the first SPH fluid simulations designed to run entirely and efficiently on the GPU [HKK]. This is accomplished by introducing a 3 dimensional grid to sort the particles into, before searching for neighbors. More recent implementations typically use a variation of counting sort as described by Hoetzlein to accomplish the same task [Hoe14]. Since Harada *et al.*'s work, there has been a lot of research dedicated to improving SPH, such as the work done by Yan *et al.* which introduced an adaptive surface tension model and a modified pressure equation, among other things [YWH*09].

This work proposes a GPU-based SPH fluid simulation using counting sort for optimizing neighbor search. It is based-off 3 SPH implementations, and a GPU sorting algorithm, as outlined in the following list.

- Müller *et al.* 2003 which proposes the first SPH simulation to acheive interactive frame rates [MCG03].
- Harada *et al.* 2007 which proposes the first entirely GPU-based SPH simulation [HKK].
- Yan *et al.* 2009 which proposes a number of improvements to SPH, among of which is a modified pressure equation. [YWH*09]
- Hoetzlein 2014 which proposes a GPU particle sorting algorithm similar to radix sort, called counting sort [Hoe14].

3. Overview

This section will give an overview of the ideas behind this project, as well as the implementation. The first subsection will outline the theory behind this simulation including the governing equations and logic behind the neighbor search. The second subsection will discuss how these concepts were implemented.

3.1. Theory

This section describes the ideas used to build this simulation. It will start by discussing what SPH actually is, next how it can be used to simulate fluids, next the ideas behind the neighbor search, and finally the boundary conditions used.

3.1.1. Smoothed Particle Hydronamics

Smoothed Particle Hydronamics is a method for interpolating values between particles in arbitrary spaces. To do this, SPH computes weighted sums of nearby particle contributions using radial symmetric smoothing kernels [MCG03]. The smoothing kernel has a radius at which it will evaluate to 0 for all values beyond it. Particles within each other's kernel radius, influence each other's behaviour. As Müller *et al.* note, since the gradient and laplacian operators only affect the kernel function, SPH cannot gaurentee common physical principals like symmetry of forces and conservation of momentum [MCG03]. Nevertheless, when implemented carefully SPH can produce results that look convincing *enough*.

3.1.2. Simulating Fluids with SPH

Fluid motion is defined by the Navier Stokes equations which declare conservation of momentum and mass, respectively.

$$\dot{u} + (\nabla \bullet u)u = \frac{du}{dt} = -\frac{1}{\rho}\nabla p + v\nabla^2 u + g \tag{1}$$

$$\nabla \bullet u = 0 \tag{2}$$

Where u is velocity, ρ is density, p is pressure, v is the viscosity coefficient, and g is external forces (i.e. gravity). Since SPH uses particles with a constant mass, mass conservation is guarenteed and so equation (2) can be ignored. Equation (1) gives the momentum of the fluid. On the right hand there are three forces contributing to the fluid's motion modeling pressure ($-\frac{1}{\rho}\nabla p$), viscosity ($v\nabla^2 u$), and external forces (g).

This section defines the equations used to simulate fluid with SPH. The weighted sums compute a property for position x and will be of the form

$$\phi(x) = \sum_j m_j \frac{\phi_j}{\rho_j} W(x - x_j) \tag{3}$$

where m_j, ρ_j, and x_j are particle j's mass, density, and position, respectively. W is the kernel function and is different depending on the property being computed. The density of a particle with position x is computed using Harada *et al.*'s method.

$$\rho(x) = \sum_j m_j W_{poly6}(x - x_j) \tag{4}$$

With density, pressure is typically computed using the ideal gas equation.

$$p = k(\rho - \rho_0) \tag{5}$$

where k is the gas constant and ρ_0 is rest density. Instead, the equation proposed by Yan *et al.* to better enforce incompressibility is used.

$$p = k\left(\left(\frac{\rho}{\rho_0}\right)^3 - 1\right) \tag{6}$$

The pressure and viscosity forces for particle i are computed as F_i^{press} and F_i^{vis}, as is done by Harada *et al.*

$$F_i^{press} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W_{spiky}(x_i - x_j) \tag{7}$$

$$F_i^{vis} = v \sum_j m_j \frac{v_j - v_i}{\rho_j} \nabla W_{vis}(x_i - x_j) \tag{8}$$

The same kernel functions used by Müller *et al.* are used, given as follows.

$$W_{poly6}(r_{ij}) = \frac{315}{64\pi h^9}(h^2 - |r_{ij}|^2)^3 \tag{9}$$

$$\nabla W_{spiky}(r_{ij}) = -\frac{45}{\pi h^5}(h - |r_{ij}|)^2 \frac{r_{ij}}{|r_{ij}|} \tag{10}$$

$$\nabla W_{vis}(r_{ij}) = \frac{45}{\pi h^6}(h - |r_{ij}|) \tag{11}$$

where h is the kernel radius, and every kernel function is 0 for $|r_{ij}| \geq h$.

3.1.3. Nearest Neighbor Search

Since many of the equations from the previous section involve a weighted sum of neighbor particles, neighbors for every particle must be found at every timestep. This is a well known bottleneck of SPH fluid simulations [HKK]. To solve this problem, a grid-based approach can be used, similarly to how Eulerian methods use semi-Lagrangian advection. If the space is divided into a grid, particles can be placed into discrete cells. If a cell is no smaller than the kernel radius, then to find all relevant particle neighbors, only neighboring bins need to be searched instead of all particles in the simulation. This means searching a maximum of 9 cells in 2D and 27 in 3D. There are different ways of performing this sort, but some common methods used on the GPU are radix and counting sort [Hoe14].

3.1.4. Boundary Conditions

Instead of using the wall weight function defined by Harada *et al.*, Euler impulse response collisions are used to simplify this projecet. While that stops particles from going out of bounds, they may still stick to walls since they perceive a low density there. To remedy this slightly, some density is added to a particle that has a wall within it's kernel radius. This is based of the idea of wall particles, but not as strong as it could be if it were more carefully thought out. Instead of computing the density for all the possible wall particles within the kernel radius, density is added for a single particle scaled by 4.

3.2. Implementation

This section will explain how the previously described concepts are implemented to create a performant GPU-based fluid simulation. This system is implemented with the Cinder Creative Coding Toolkit, a C++ framework built on OpenGL. To run this simulation on the GPU, compute shaders that read and write from Shader Storage Buffer Objects (SSBOs) are used. To represent each particle, a struct is used containing velocity, density, position, and pressure. These values are combined into a single struct to simplify reording during sorting, as opposed to Harada *et al.*'s implementation where each property is in a separate buffer. In total, the system uses 6 buffers, 2 for particles, 2 for count and offset, plus 2 for debugging. Figure 1 shows the simulation's high-level flow. The sorter reorders particles by bin, next density and pressure are computed, finally forces are found and applied to each particle. The sorter takes particles (1) as input and outputs to particles (2), the density program reads and writes from this buffer, the update program then takes particles (2) as input and outputs bak to particles (1). The following subsections will explain the implementations of the particle sorter, SPH fluid simulator, and particle renderer.

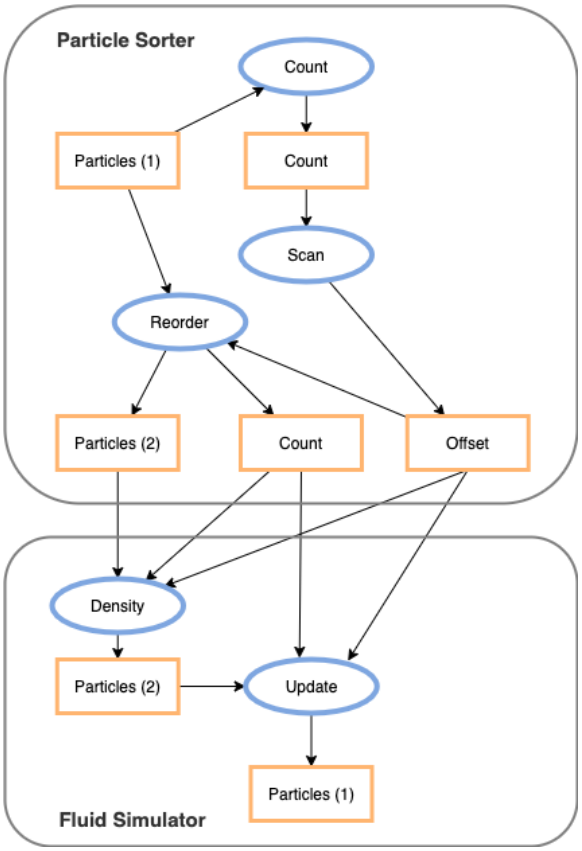

Figure 1: *Flow diagram of the implementation. Ovals are compute shaders and rectangles are buffers. Note that while some buffers appear twice, they refer to the same memory, and are merely duplicated to simplify the diagram.*

3.2.1. Particle Sorter

To sort particles into bins, a series of compute shaders are used that take the original particle buffer as an input and outputs a reordered one, along with a count and offset buffer used to look up particles by bin.

The count program computes a particle's bin index, and then increments an SSBO using an atomic add operation. This can be used instead of a typical radix count because the order within each bin does not matter, so there is no need to preserve the initial ordering [Hoe14].

The scan program computes the offset (i.e. the number of particles before the first particle in a bin) for each bin. This is probably the bottleneck of this implementation, since it is done iteratively within a single shader invocation. A linear scan can be trivially implemented in a shader the same way it would be in C++, simply by looping through bins and computing the prefix sum along the way. Rewriting this to use a parallel scan as described by Harada and Howes could greatly improve the performance of the system [HH11].

The last step of the sorter is to reorder particles based on bin. First, the count buffer is cleared, then another count is performed by the reorder program, this time inserting particles into the output buffer based on the previously computed offset and the newly (atomically) computed count. This works because an atomic counter returns the previous value, which in this context is the local bin offset for a particle. The order may be different than the previous count, but as stated earlier, order within each bin doesn't matter. The final particle buffer, along with the count and offset buffers are returned to the simulation for use when computing fluid motion.

3.2.2. SPH Fluid Simulator

Once particles are sorted by bin, computing properties for each particle based on each neighbor becomes a lot less expensive. A small price to pay for this is that offset and count buffers must be read from, but they can be qualified as "readonly" to optimize parallel reads.

The density program takes particles, offset, and count. For a given particle it searches neighboring bins for particles within the kernel radius, computing the summation in Equation (4), including any wall particles. Next, equation (6) is used to compute pressure. These properties are written back to the input buffer, which is safe because the computation does not use particle density or pressure, only position and mass (which are constant during this step).

The update shader is the where most of the simulation work happens. First, pressure and viscosity forces are computed in the same way as density, but computing equations (7) and (8). The resulting forces are summed together with a gravity force to define the particle's momentum as is defined by equation (1). Dividing by particle density gives acceleration and with this, Euler integration is used to compute the next velocity and position. Before writing the new values to the output buffer, boundaries are enforced. If a parrticle is out of bounds, an Euler collision is applied with strong damping.

3.2.3. Particle Renderer

Particles are rendered as spheres using "GL_POINTS" and a sphere fragment shader. The color of each particle depends on both pressure and speed, weighted differently for each RGB channel. The weighting produces a gradient of colors from dark blue, turqoise, light blue, to nearly white. This configuration results in particles at the surface appearing lighter, similar to waves one might see on a river, lake, or ocean.

4. Evaluation

This section is broken into two subsections that will describe the qualitative and quantitative results of the implementation.

4.1. Qualitative Results

The demo used for qualitative analysis is 80,000 particles initialized in a dam break configuration where the particles form a cube in the corner. The simulation runs almost completely smoothly producing about 60 frames per second. Figure 2 shows a still from this demo highlighting 4 qualities of the simulation. While there are certainly problems, overall the results look good and behave fluid-like.

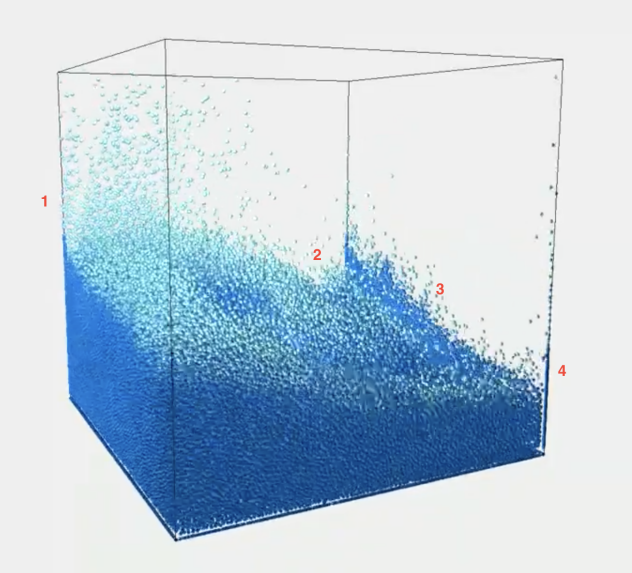

Figure 2: *A still from an experiment with 80,000 particles initialized as a cube dam break, annotated to highlight simulation qualities (1) a breaking wave, (2) the viscous motion of the fluid, (3) particles sitcking to and stacking on the wall, (4) particles sticking to the corner.*

The breaking wave (1) in Figure 2 shows how the simulation capture this behaviour commonly seen in natural bodies of water. The particles here have a low pressure and high velocity so they are rendered as a gradient from light blue to white.

Between (1) and (2) the gradient is from light blue to turquoise and deep blue, showing different balances of pressure and velocity. This part of the image shows the viscosity force holding particles together forming curling waves.

(3) shows a group of particles getting stuck to and stacking on the wall. These particles behave almost like a 2D SPH fluid simulation, but eventually do come off the wall. This is happening because boundary particles are not properly implemented, only pseudo boundary particles and Euler collisions are used. This would probably be one of the most important improvements to make if work on the system were to continue.

(4) highlights a couple related problems to (3). They are related because they are most likely caused by the same problem, and also would probably be fixed if boundary particles were properly implemented. Along the vertical corner particles collect and create a spring-like motion. Lower down, below the water level, there is a space unoccupied by particles between the main group of particles and the edge particles. This space can also be seen along the floor edges. With only Euler collisions implemented, particles near the wall will sense a very low density due to a lack of particles. They then stick to the wall but repel the body of particles within the boundaries. The psuedo wall particle helps stop particles from stacking up along the walls, but it does not completely solve it and does not help with the corners or particles along the edge within the body. This is probably because the pseudo wall particle density is no where near as strong as it should be, so it has no chance of counteracting the density of the fluid body.

4.2. Quantitative Results

The experiments were performed on a 2019 16" MacBook Pro, with a 2.3 GHz 8-Core Intel Core i9 processor, 32 GB of RAM, and an Intel UHD Graphics 630 1536 MB graphics card, running Windows 10. Figure 3 illustrates the results of a performance test using 100,000-400,000 particles. As the graph shows, while the system can handle 100,000 stably at 60 frames per second, performance quickly drops off when pushing beyond that. Furthermore, while the graph does not relfect this, when there are any more than 200,000 particles, the simulation becomes unstable and no long behaves as espected. While there is a definite ceiling of scalability in this system, it is sufficiently high since 100,000 particles is more than enough to produce interesting results. It is worth noting, that as Harada *et al.* stated, the search is key to the performance of this system. To illustrate this, the system's performance hits a ceiling around 20,000 particles without sorting. The improvements sorting provides are clear since the final system can handle 5 times as much particles with the same performance. Since the system uses a linear scan, there are even more performance gains to be had with a parallel scan. Implementing a parallel scan would be one of the most important things to do to improve system's performance.

5. Challenges

Implementing this system was not easy and many difficult challenges presented themselves along the way.

First, debugging on the GPU is quite a challenge of its own. Eventually systems were devised to make this easier, but earlier in the project this was a major problem because it was difficult to know what was working and what wasn't. As the project developed, utility functions for reading data back to the CPU from a GPU buffer as well as shaders programs for rendering simulation data as points were created to make debugging easier.

Another major challenge was the fact that it seemed impossible to write to a 3D texture from a compute shader in the environment used. This was the source of a lot of confusion and debugging. No matter what, it seemed writing to a 3D texture was not an option so everything was done with SSBOs. Why 3D textures did not work is still unknown, it may even be due to drivers or Cinder. Eventually this was realized and textures were completely avoided, but unfortunately a lot of time was wasted to this earlier on in the project.

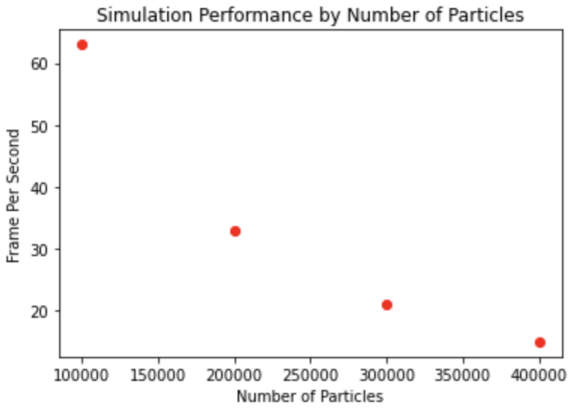

Figure 3: *A comparison of the systems performance in frames per second when simulating different numbers of particles. This figure helps illustrate the performance limits of the system.*

Even once the code is functioning, SPH can be so delicate that it can be hard to know that the system is working correctly. Often, changing a single configuration parameter may cause the entire system to break, or change the behaviour entirely. Without fully understanding the delicacy of parameters, it may be easy to think the system suddenly broke, causing a wasteful debugging session.

Lastly, particle behaviour near boundaries is an ongoing challenge that has been remedied somewhat but has not been completely solved. While this project was full of very difficult challenges, the end result is a successful simulation that produces mostly desired results.

6. Future Work

While this system successfully simulates fluid using SPH on the GPU, there are many improvements to be made. As mentioned earlier, one of the main improvements this system needs is a parallel scan.

Second, the system's boundary conditions need to be updated to properly implement wall particles as described by Harada *et al.* This should solve some of the main problems with the system, particles sticking to walls, forming springs in corners, and there being a gap between wall and fluid body particles.

Next, the behaviour could be improved by introducing surface tension forces as described by Yan *et al.*. In SPH fluid simulations, particles at the surface observe a low pressure when in reality these particles should oberse a higher pressure due to surface tension forces present in real-world fluids. This can be observed by carefully filling a glass just slighlty too full and noticing that while the water level surpasses the cup's rim, the water does not spill. This kind of behaviour can be captured by including surface tension forces to the simulation.

While the particles in this system are colored in such a way that the rendered result resembles water, it is still particles rather than a smooth surface like real water. Surface reconstruction is always the last step in a fluid simulation and it can be done in a variety of ways. Two common ways to do this are marching cubes and ray marching. Once the behaviour of this system is improved, adding surface reconstruction would add a new dimension to the system by rendering something that looks like an actual fluid as opposed to a collection of particles.

7. Conclusion

This project verifies Harada *et al.*'s work by showing that their design can be implemented to create an efficient SPH fluid simulation on the GPU. It also verifieded Hoetzlein's work by showing that their algorithm can be implemented to create an efficient nearest neighbor search that can improve the speed of an SPH fluid simulation by 5 times.

References

[BGOS06] BARGTEIL A. W., GOKTEKIN T. G., O'BRIEN J. F., STRAIN J. A.: A semi-lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics (TOG) 25*, 1 (2006), 19–38. 1

[DG96] DESBRUN M., GASCUEL M.-P.: Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation'96*. Springer, 1996, pp. 61–76. 1

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), pp. 23–30. 1

[FM96] FOSTER N., METAXAS D.: Realistic animation of liquids. *Graphical models and image processing 58*, 5 (1996), 471–483. 1

[Har62] HARLOW F. H.: *The particle-in-cell method for numerical solution of problems in fluid dynamics*. Tech. rep., Los Alamos Scientific Lab., N. Mex., 1962. 1

[HH11] HARADA T., HOWES L.: Introduction to gpu radix sort. *Heterogeneous Computing with OpenCL. Morgan Kaufman* (2011). 4

[HKK] HARADA T., KOSHIZUKA S., KAWAGUCHI Y.: Smoothed particle hydrodynamics on gpus (2007). *URL http://inf. ufrgs. br/cgi2007/cd_cgi/papers/harada. pdf*. 2, 3

[Hoe14] HOETZLEIN R. C.: Fast fixed-radius nearest neighbors: Interactive million-particle fluids.(2014). *URL http://on-demand. gputechconf. com/gtc/2014/presentations/S4117-fast-fixed-radius-nearest-neighbor-gpu. pdf* (2014). 2, 3, 4

[HW65] HARLOW F. H., WELCH J. E.: Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids 8*, 12 (1965), 2182–2189. 1

[KO96] KOSHIZUKA S., OKA Y.: Moving-particle semi-implicit method for fragmentation of incompressible fluid. *Nuclear science and engineering 123*, 3 (1996), 421–434. 1

[LSSF06] LOSASSO F., SHINAR T., SELLE A., FEDKIW R.: Multiple interacting liquids. *ACM Transactions on Graphics (TOG) 25*, 3 (2006), 812–819. 1

[MCG03] MÜLLER M., CHARYPAR D., GROSS M. H.: Particle-based fluid simulation for interactive applications. In *Symposium on Computer animation* (2003), pp. 154–159. 1, 2

[Mon92] MONAGHAN J. J.: Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics 30*, 1 (1992), 543–574. 1

[REN*04] RASMUSSEN N., ENRIGHT D., NGUYEN D., MARINO S., SUMNER N., GEIGER W., HOON S., FEDKIW R.: Directable photorealistic liquids. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), pp. 193–202. 1

[Sta99] STAM J.: Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), pp. 121–128. 1

[YWH*09] YAN H., WANG Z., HE J., CHEN X., WANG C., PENG Q.: Real-time fluid simulation with adaptive sph. *Computer Animation and Virtual Worlds 20*, 2-3 (2009), 417–426. 2