Interactive Air-Water-Fluid Interactions Using SPH

WILLIAM TREMBLAY, McGill University

This paper presents a Rust implementation of a SPH (Smoothed Particle Hydrodynamics) model extended to model complex interactions between different fluids of different types and air. The model can simulate (im)miscibility of fluids due to polarity, temperature diffusion, effects of temperature on density, and trapped air pockets inside fluids. The implementation is multi-threaded, makes use of a cache-friendly spatial hashing approach, and dynamically generates air particles only when needed. This allows interactive simulations with up to tens of thousands of particles at fine time-steps.

Code: https://github.com/tremwil/rust_sph
Video: https://youtu.be/Yw7QlvvZDig

Additional Key Words and Phrases: fluid simulation, smoothed particle hydrodynamics, two dimensional, fluid-fluid interaction, air-fluid interaction, optimization, Rust

ACM Reference Format:
William Tremblay. 2023. Interactive Air-Water-Fluid Interactions Using SPH. 1, 1 (April 2023), 4 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Interactions between multiple fluids, especially immiscible ones, can generate captivating emulsion patterns that can be particularly visually appealing, such as those created by lava lamps. However, reproducing such phenomena using computer simulation appears to be a rather complex endeavor, especially when one attempts to capture all particularities of such interactions. Hence we consider the use of smoothed particle models (SPH) to approximate these phenomena so that they can be simulated at interactive rates.

2 RELATED WORK

The main ideas behind this SPH implementation come from [Müller et al. 2003] and [Müller et al. 2005]. The former describes a simple, single-fluid SPH model which is used as a basis for the multi-fluid model developed in [Müller et al. 2005]. Some modifications to these methods have nonetheless been made, and will be discussed in the following section.

3 METHODS

3.1 Base SPH Model

The fundamental idea behind this multiple-fluid model is the single-fluid SPH model described by [Müller et al. 2003]. Their method involves discretizing a property α of the fluid by encoding said property into a finite set of *particles* indexed $1, \dots, n$, and approximating its value using a *smoothing kernel* $W(\vec{r}, h)$:

$$\alpha(\vec{x}) = \sum_{j=1}^{n} \alpha_j \frac{m_j}{\rho_j} W(\vec{x} - \vec{p}_j, h) \tag{1}$$

Author's address: William Tremblay, McGill University, william.f.tremblay@mail.mcgill.ca.

2023. XXXX-XXXX/2023/4-ART $15.00
https://doi.org/0000001.0000001

where m_j, ρ_j and $\vec{p}_j$ are the mass, density and position of particle j, respectively, and h is the *interaction*, or *sampling* radius. Since the gradient, Laplacian and divergence of the above equation depends only depend on that of W by linearity, one can write, say, the Navier-Stokes equations under an SPH formulation. Doing so, [Müller et al. 2003] obtain, for particle i:

$$\rho_i = \sum_{j=1}^{n} m_j W(\vec{p}_i - \vec{p}_j, h) \tag{2}$$

$$f_i^{\text{pressure}} = -\sum_{j=1}^{n} p_j \frac{m_j}{\rho_j} \nabla W(\vec{p}_i - \vec{p}_j, h)$$

$$f_i^{\text{viscosity}} = \mu \sum_{j=1}^{n} \vec{v}_j \frac{m_j}{\rho_j} \nabla^2 W(\vec{p}_i - \vec{p}_j, h)$$

where p_j is the pressure of particle j, which the authors compute as $k(\rho_J - \rho_0)$ for some constant k and rest density of the fluid ρ_0, μ is the fluid's viscosity, and $\vec{v}_j$ is the velocity of particle j. However, one might note that the f_i above are not symmetric; as such the authors replace them with the following:

$$f_i^{\text{pressure}} = -\sum_{j=1}^{n} \frac{p_i + p_j}{2} \frac{m_j}{\rho_j} \nabla W(\vec{p}_i - \vec{p}_j, h) \tag{3}$$

$$f_i^{\text{viscosity}} = \sum_{j=1}^{n} \mu \frac{\vec{v}_j - \vec{v}_i}{2} \frac{m_j}{\rho_j} \nabla^2 W(\vec{p}_i - \vec{p}_j, h) \tag{4}$$

These equations transfer directly to the multiple fluid case, except for (3) in which [Müller et al. 2005] let

$$\mu = \frac{\mu_i + \mu_j}{2}$$

so that symmetry holds.

The original model uses a leapfrog integration scheme, but we have found sympletic Euler to be sufficient while requiring less information to be stored inside the particle objects.

3.2 Collision Detection and Resolution

For collision detection, a straightforward approach similar to that described by [Müller et al. 2003] is used, which involves computing the axis of least separation of the particle's position with the boundary object, pushing it out, and reflecting its velocity along the axis. However, in this paper we instead compute a radius for the particle as follows:

$$r_i = \frac{1}{2}\sqrt{\frac{m_i}{\rho_i}}$$

and resolve the circle collision instead. This does not appear to affect the visual accuracy of the simulation, but it does avoid rendering artifacts caused by particles pushed completely against the surface.

3.3 Interface and Surface Tensions

To simulate attraction and repulsion forces due to polarity differences, we follow [Müller et al. 2005] and define two additional scalar

fields c^s and c^i on the particles. c^s is set 1 for all liquid particles and thus encodes liquid-air boundaries, while c^i is set to $\pm 1/2$ depending on the polarity of the particle to encode boundaries between immiscible liquids. The interface and surface forces are then given by

$$f_i^{\text{interface}} = -\sigma^i \nabla^2 c^i \frac{\nabla c^i}{|\nabla c^i|}, \qquad f_i^{\text{surface}} = -\sigma^s \nabla^2 c^s \frac{\nabla c^s}{|\nabla c^s|}, \tag{5}$$

where the gradients and laplacians are computed by linearity according to Eq. (1).

3.4 Heat Diffusion

The heat diffusion model is very simple and follows [Müller et al. 2005] exactly. The heat equation being $u' = c\nabla^2 u$ for some constant c, applying Eq. (2) we get

$$u_i' = c \sum_j m_j \frac{u_j - u_i}{\rho_j} \nabla^2 W(\vec{p}_i - \vec{p}_j, h).$$

We integrate this using the Forward Euler method and then let the particle's rest density, ρ_0, be inversely proportional to the temperature.

3.5 Trapped Air Generation

The presence of air pockets, or "bubbles" in an agitated fluid is an important part of what make a fluid simulation look realistic. However, fully simulating all air particles around the fluid is prohibitively computationally expensive. Hence we muse the method for [Müller et al. 2005] which boils down to the following conditions,

- If $|\nabla c^p| > t_p$ (at liquid-vacuum boundary) and $\nabla c^p \cdot \vec{g} < 0$ (liquid surface is facing downwards), then spawn a particle at $x - d\nabla c^p$.
- If we have $|\nabla c^s| > t_s$ (not at liquid-air boundary) and $|\nabla c^p| > t_p$ (at liquid-vaccum boundary), or if $\rho < t_\rho$ for an air particle, delete it.

Here, c^p is a smoothed variable which is 1 for all particles, and t_p, t_s, t_ρ all hand-tuned parameters. Additionally, an artificial buoyancy force proportional to $(\rho - \rho_0)\vec{g}$ is applied to the air particles to offset the pressure overestimate caused by low air particle counts, and allow it to rise to the surface.

3.6 Smoothing Kernels

In the past sections the smoothing kernel W has appeared in many equations. Following [Müller et al. 2003], we use 3 different kernels depending on what fluid property is being sampled. The paper provides these kernels for 3D; however, [Vijaykymar 2012] gives their equivalent in two dimensions:

$$W_{poly}(\vec{r}, h) = \frac{4}{\pi h^8}(h^2 - ||\vec{r}||^2)^3,$$
$$W_{spiky}(\vec{r}, h) = \frac{10}{\pi h^5}(h - ||\vec{r}||)^3,$$
$$W_{viscosity}(\vec{r}, h) = \frac{10}{3\pi h^2}\left(-\frac{||\vec{r}||^3}{2h^3} + \frac{||\vec{r}||^2}{h^2} + \frac{h}{2||\vec{r}||} - 1\right).$$

[Müller et al. 2003] uses W_{poly} for density, interface tension and heat diffusion, but in 2D this kernel's sign-switching Laplacian appears to be a much bigger issue and leads to heat values diverging. This may be because the "volume" stored in the outer ring where this sign change occurs is more significant due to the reduced dimensionality. Hence after some experimentation, this implementation uses W_{spiky} for density and pressure calculations, $W_{viscosity}$ viscosity and heat diffusion, and W_{poly6} for interface forces.

3.7 Optimizations

To speed up the simulation, we use two ideas: spatial hashing and parallelism. The spatial hash data structure used is given below:

```
  /// Spatial hash grid cell.
#[derive(Default, Clone)]
pub struct GridCell
{
    /// Set of particles which currently lie in this grid cell.
    pub particles: HashMap<ParticleId, FluidParticle>,
    /// Boundaries, stored in the cell to accelerate collisions.
    pub boundaries: Vec<usize>,
    /// Interacting bodies, stored here to accelerate collisions.
    pub interacting_bodies: Vec<usize>,
}
#[derive(Clone)]
/// Spatial hash data structure, meant to accelerate range queries.
pub struct SpatialHash {
    cells: HashMap<IVec2, GridCell>,
    particle_cells: HashMap<ParticleId, IVec2>,
    step: f32,
    inv_step: f32,
}
```

The hash map data structure used is from [d'Antras et al. 2023] and is extremely performant. Since it is not node based and instead uses quadratic probing, it boasts excellent cache performance. Hence the `FluidParticles` are stored directly inside the `GridCells` (instead of behind a pointer) to make use of this when performing neighbor queries. Indices of boundaries and other objects with collision (particle sinks and temperature sources) are stored inside grids as well to optimize collision detection.

To multithread the simulation without sacrificing determinism, the solid `Rayon` crate from [Stone 2023] is used. The physics update step is split into 3 parts:

Density Update. Densities are updated according to Eq. (2). This is done in parallel with threads splitting the work over an equal amount of particles, then performing fast neighbor queries using the spatial hash.

Smoothed Property Computations. All smoothed fluid properties, such as pressure/viscosity/interface forces, color field gradients, etc., are computed at once per particle, with the work again being split over threads and making use of the spatial hash.

Integration and Other Interactions. Particle positions and velocities are updated from the previously computed forces, non-fluid heat diffusion is applied, air particles are generated and stale ones are deleted. This is all done in a single thread, but is $O(n)$ and does not appear to be a bottleneck unless the number of particles is low. Care is taken to minimize the number of unnecessary memory (de)allocations as particles are dynamically created and destroyed during this step.

Rendering

The rendering and UI components of the implementation make use of the `macroquad` Rust crate [Logachev 2020]. With the simulation

being capable of many tens of thousands of particles, another challenge arises, namely drawing them at a high resolution quickly. For this an idea by [Parker 2020] was ported to `macroquad`'s OpenGL layer. The author passes a single quad to the GPU and uses a fragment shader to draw up to millions of circles with pixel-precision by doing a single instanced draw call. This made the particle rendering multiple orders of magnitude faster than a naive loop over `macroquad`'s `draw_circle` function.

4 RESULTS

For performance metrics, note the following was tested on a PC with 16GB of 2400MHz DDR4 RAM, an Intel i5 12600K CPU @ 4.1GHz and an NVidia RTX 3070 graphics card.

4.1 Stability

One of the main issues with a SPH model that isn't divergence free is that to make liquids appear non-compressible, they must be assigned a very high stiffness k. This leads to large pressure forces that require careful either aggressive damping via high viscosities or taking small timesteps. In our tests, we have found that a viscosity of 1 and stiffness of 50 produced visually appealing results, even though water's real viscosity is two orders of magnitude lower. For most examples, we use timesteps of 0.0005 seconds, but the simulator can usually ensure stability up to 0.002 seconds or more depending on particle size and count.

4.2 Multiple-Fluid Interactions

Blobs of water ($1000kg/m^2$) and oil ($900kg/m^2$) were dropped into ethanol ($789kg/m^2$). Both water and ethanol were modeled as polar, while oil was made non-polar, leading to emulsion at the boundary. The fluids unmixed themselves due to density differences (Fig. 1).

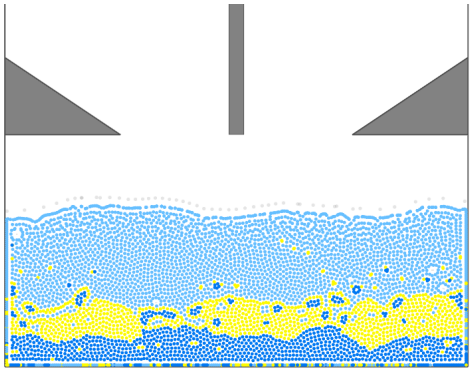

Fig. 1. Oil, water and EtOH are mixed together.

4.3 Trapped Air

In this test, an emitter generates water particles at a rate of 3000/s, while a sink under the circle destroys them. The particles collide with the circle and form droplets that fall in the pool of fluid below, creating many air bubbles (white) (Fig 2).

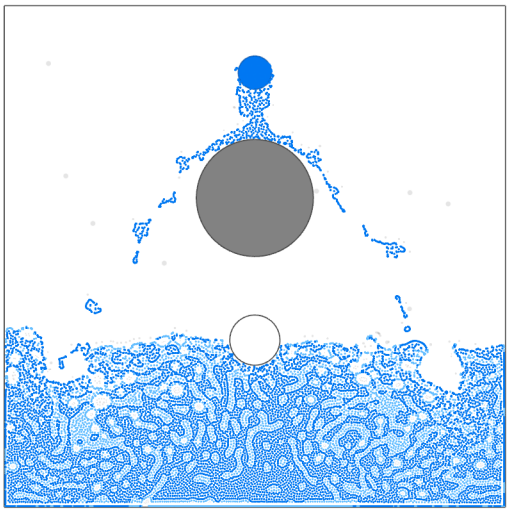

Fig. 2. Fluid droplets fall and create bubbling.

4.4 Heat Diffusion

Heat and cold sources (set to 500K and 100K respectively) were placed at each end of the fluid to create a temperature gradient. The fluid density changed as its temperature did, leading to buoyancy forces, that initiated a current in the fluid (Fig. 3). This test used 3306 particles and ran at 50% real time, with a timestep of 1/1440s.

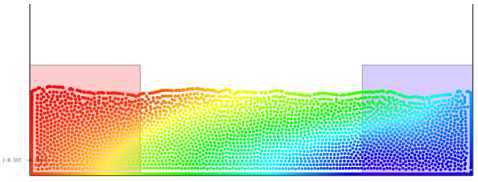

Fig. 3. A temperature gradient induces a current inside the fluid.

4.5 Performance

Three test scenes have been created to assess the model's performance. In the first, a 4585-particle dam-break scenario is played at 1/3 real-time speed at 60 frames per second, using a time step of 1/180 with 7 substeps. Hence each physics update took $1/420 \approx 2.3$ milliseconds. The second involves two 20,000 1mm fluid blocks, one of water and one of oil, colliding into one another at a relative speed of 2 m/s (Fig. 4). Oil density was set to half that of water, which is set to $1000kg/m^2$. Trapped air bubbles can also be seen. The last scene is a 100 thousand particle stress test, where a block of fluid moving to the right impacts a sphere and then a ball (Fig. 5). Timestep was set to 0.0005 seconds, and the simulation ran at 10 frames per second, or about 100ms per physics update.

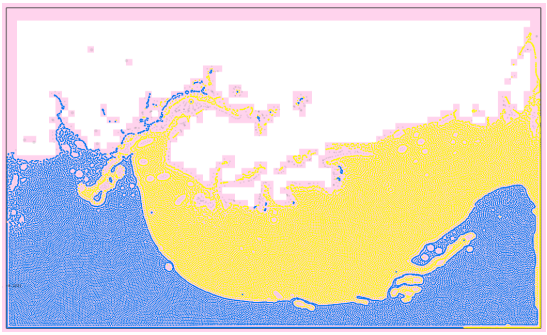

Fig. 4. Result of two 20K cubes of fluid slamming into one another.

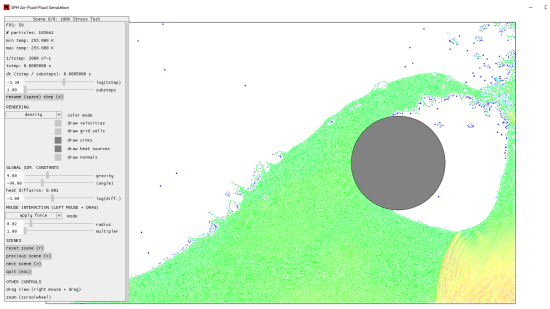

Fig. 5. 100K particle cube colliding into a sphere. Particles are colored according to their density. Generated air particles can be seen .

5 CONCLUSIONS

A high-performance Rust implementation of a multi-fluid SPH model capable of handling many different types of fluid interactions was presented. The model also accounts for trapped air particles, and is able to simulate density changes caused by heat transfer.

REFERENCES

d'Antras et al. 2023. Hashbrown: Rust port of Google's SwissTable hash map. GitHub. (2023). https://github.com/rust-lang/hashbrown

F. Logachev. 2020. Cross-platform game engine in Rust. GitHub. (2020). https://github.com/not-fl3/macroquad

M. Müller, D. Charypar, and M. Gross. 2003. Particle-Based Fluid Simulation for Interactive Applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '03)*. Eurographics Association, Goslar, DEU, 154–159. https://matthias-research.github.io/pages/publications/sca03.pdf

M. Müller, B. Solenthaler, R. Keiser, and M. Gross. 2005. Particle-Based Fluid-Fluid Interaction. *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 237–244. https://doi.org/10.1145/1073368.1073402

J. Parker. 2020. Real-time visualization of a large number of circles using OpenGL. Online. (2020). https://www.johnaparker.com/blog/circle-graphics#instanced-drawing

J. e. a. Stone. 2023. Rayon: A data parallelism library for Rust. GitHub. (2023). https://github.com/rayon-rs/rayon

A. Vijaykymar. 2012. *Smoothed Particle Hydrodynamics Simulation for Continuous Casting*. Master's thesis. KTH Royal Institute of Technology. https://www.diva-portal.org/smash/get/diva2:573583/FULLTEXT01.pdf